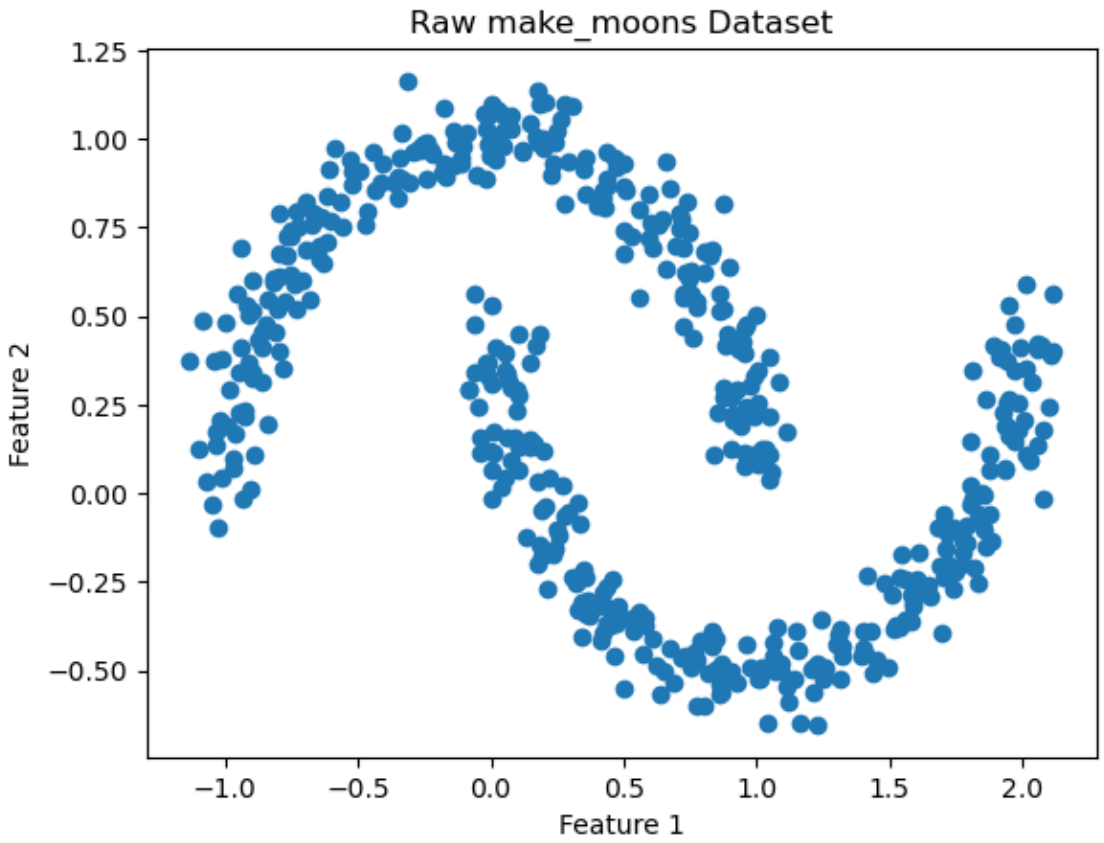
DBSCAN

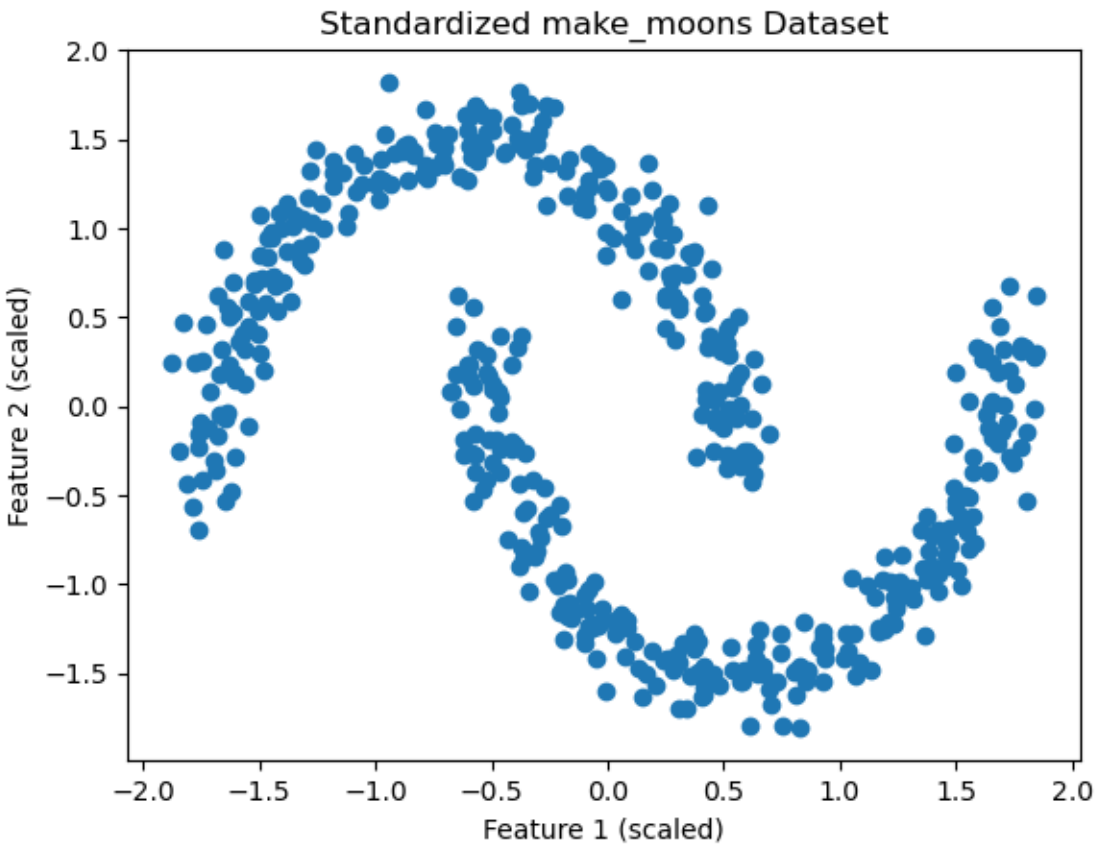
November 21, 2025

[3]:
```
# Cell 1
# Importing all required libraries for DBSCAN clustering using make_moons

import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_moons
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import DBSCAN
from sklearn.metrics import silhouette_score
```

[4]:
```
# Cell 2
# Generating the make_moons dataset and visualizing the raw data

# Create the dataset (non-linear clusters)
X, y = make_moons(n_samples=500, noise=0.07, random_state=42)

# Plot the raw dataset
plt.scatter(X[:, 0], X[:, 1])
plt.title("Raw make_moons Dataset")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()
```

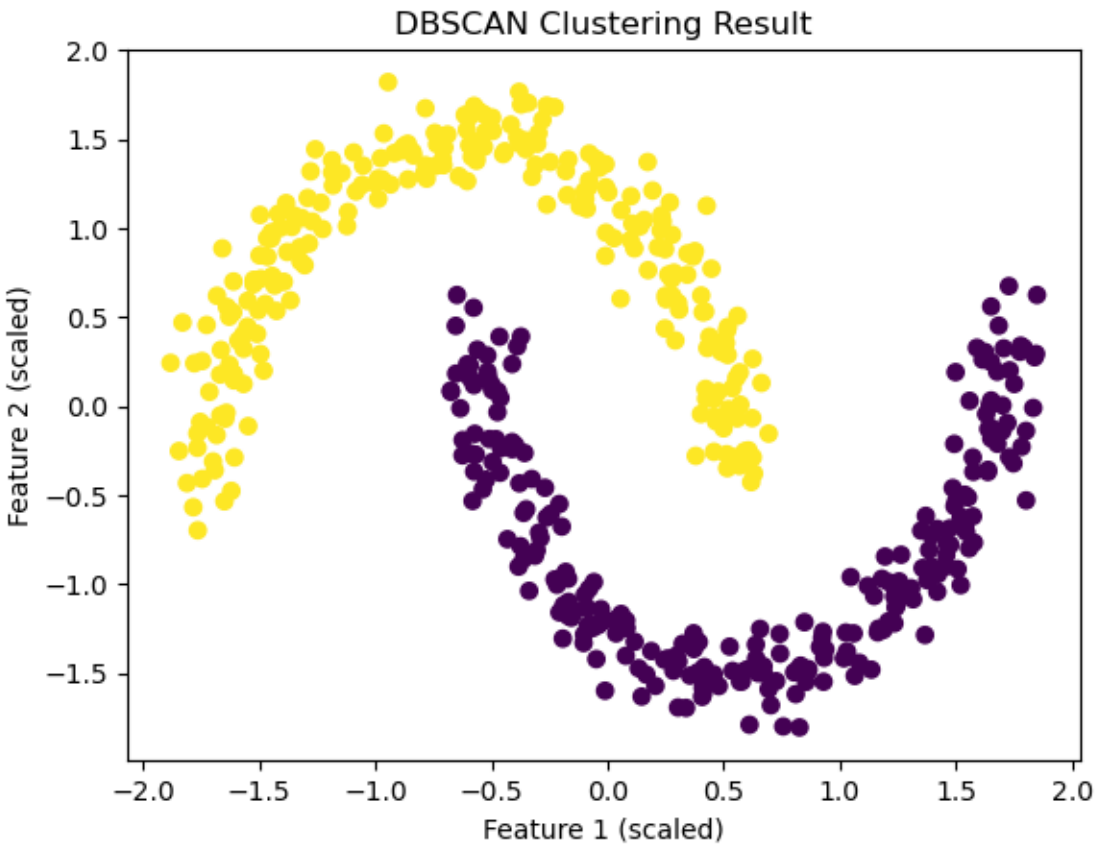


[5]:
```
# Cell 3
# Standardizing the dataset before applying DBSCAN

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Visualize the standardized dataset
plt.scatter(X_scaled[:, 0], X_scaled[:, 1])
plt.title("Standardized make_moons Dataset")
plt.xlabel("Feature 1 (scaled)")
plt.ylabel("Feature 2 (scaled)")
plt.show()
```



[6]:
```python
# Cell 4
# Applying the DBSCAN clustering algorithm

# Create DBSCAN model (default parameters can be tuned later)
dbscan = DBSCAN(eps=0.3, min_samples=5)

# Fit the model and get cluster labels
labels = dbscan.fit_predict(X_scaled)

# Print unique cluster labels (-1 means noise)
print("Cluster labels found by DBSCAN:", set(labels))

# Plot clusters using the assigned labels
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=labels, cmap='viridis')
plt.title("DBSCAN Clustering Result")
plt.xlabel("Feature 1 (scaled)")
plt.ylabel("Feature 2 (scaled)")
plt.show()
```

```
Cluster labels found by DBSCAN: {np.int64(0), np.int64(1)}
```

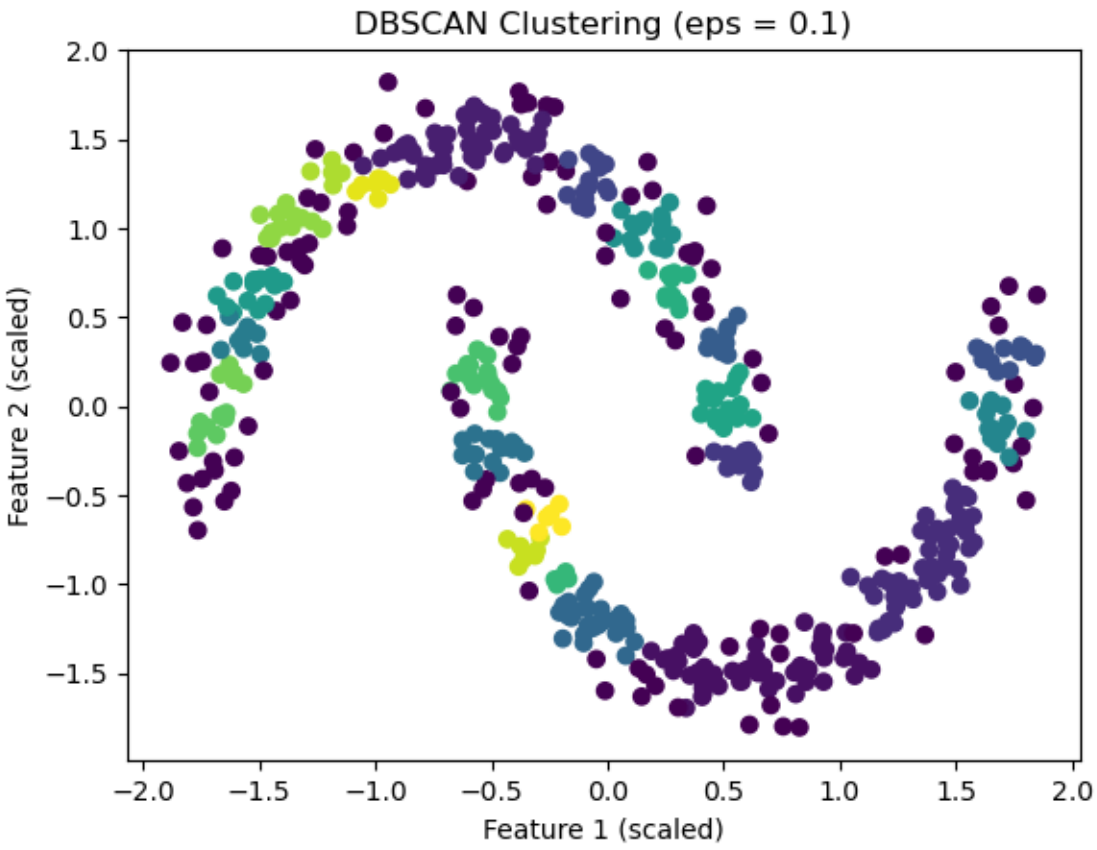


[7]:
```
# Cell 5
# Evaluating the cluster quality using Silhouette Score

unique_labels = set(labels)
print("Unique cluster labels:", unique_labels)

# Silhouette Score requires at least 2 clusters (excluding noise)
valid_clusters = [lbl for lbl in unique_labels if lbl != -1]

if len(valid_clusters) >= 2:
    score = silhouette_score(X_scaled, labels)
    print("Silhouette Score:", score)
else:
    print("Silhouette Score cannot be computed because DBSCAN found fewer than␣
 ↪2 clusters.")
```

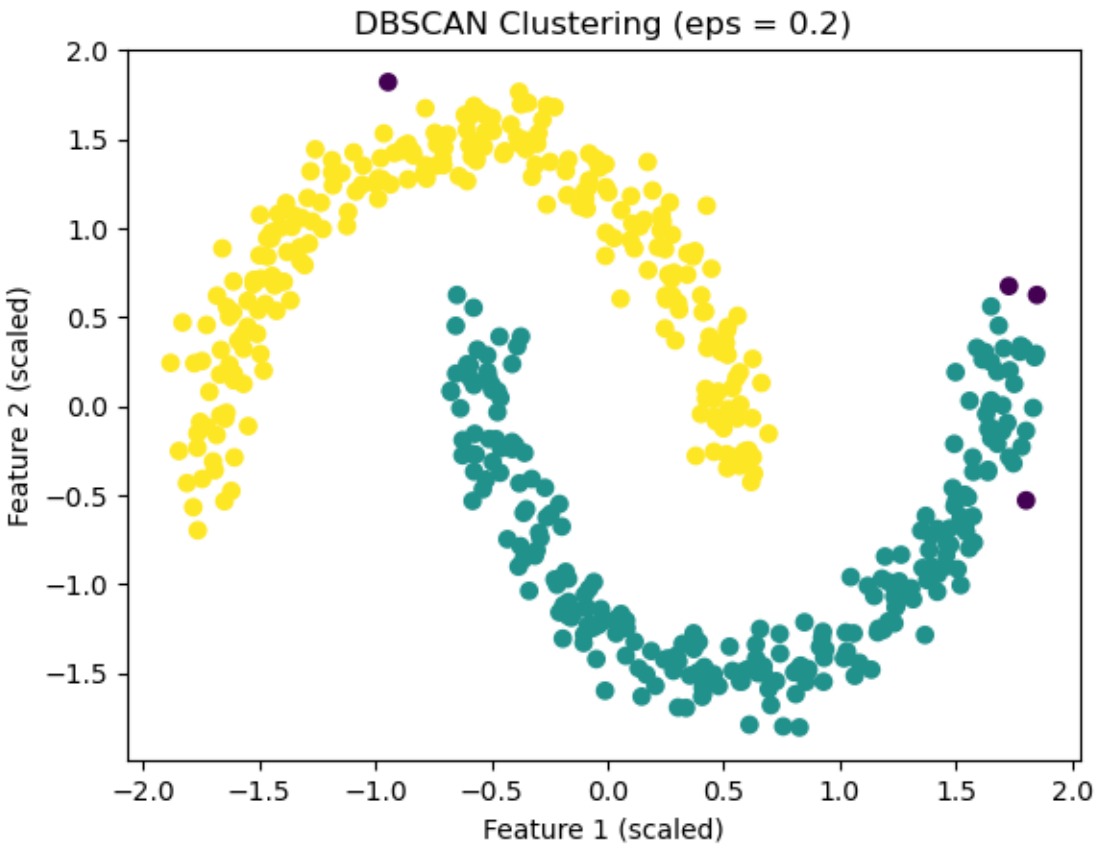
```
Unique cluster labels: {np.int64(0), np.int64(1)}
Silhouette Score: 0.38573272031287503
```

[8]:
```
# Cell 6
# Comparing DBSCAN performance for different eps values

eps_values = [0.1, 0.2, 0.3, 0.4, 0.5]

for eps in eps_values:
    db = DBSCAN(eps=eps, min_samples=5)
    labels_eps = db.fit_predict(X_scaled)

    print("\neps =", eps)
    print("Clusters found:", set(labels_eps))

    # Plot clusters for this eps value
    plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=labels_eps, cmap='viridis')
    plt.title(f"DBSCAN Clustering (eps = {eps})")
    plt.xlabel("Feature 1 (scaled)")
    plt.ylabel("Feature 2 (scaled)")
    plt.show()
```

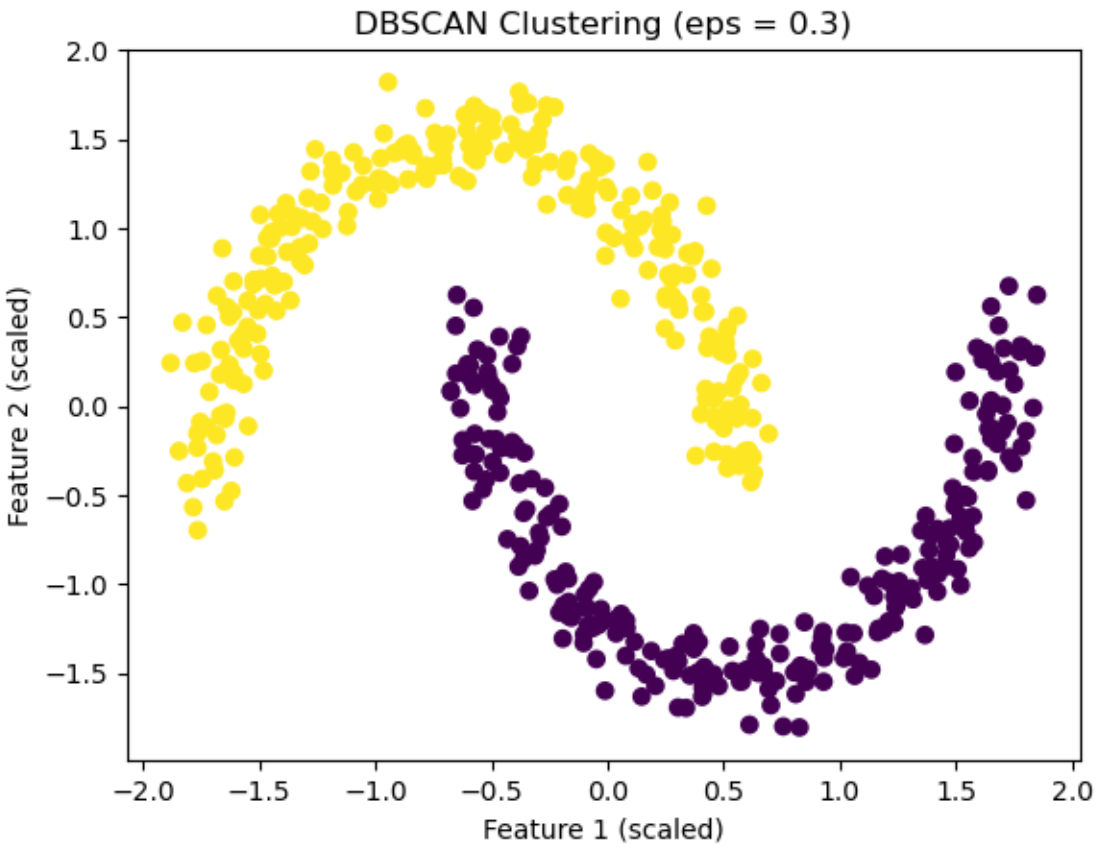
```
eps = 0.1
Clusters found: {np.int64(0), np.int64(1), np.int64(2), np.int64(3),
np.int64(4), np.int64(5), np.int64(6), np.int64(7), np.int64(8), np.int64(9),
np.int64(10), np.int64(11), np.int64(12), np.int64(13), np.int64(14),
np.int64(15), np.int64(16), np.int64(17), np.int64(18), np.int64(19),
np.int64(20), np.int64(21), np.int64(22), np.int64(23), np.int64(-1)}
```



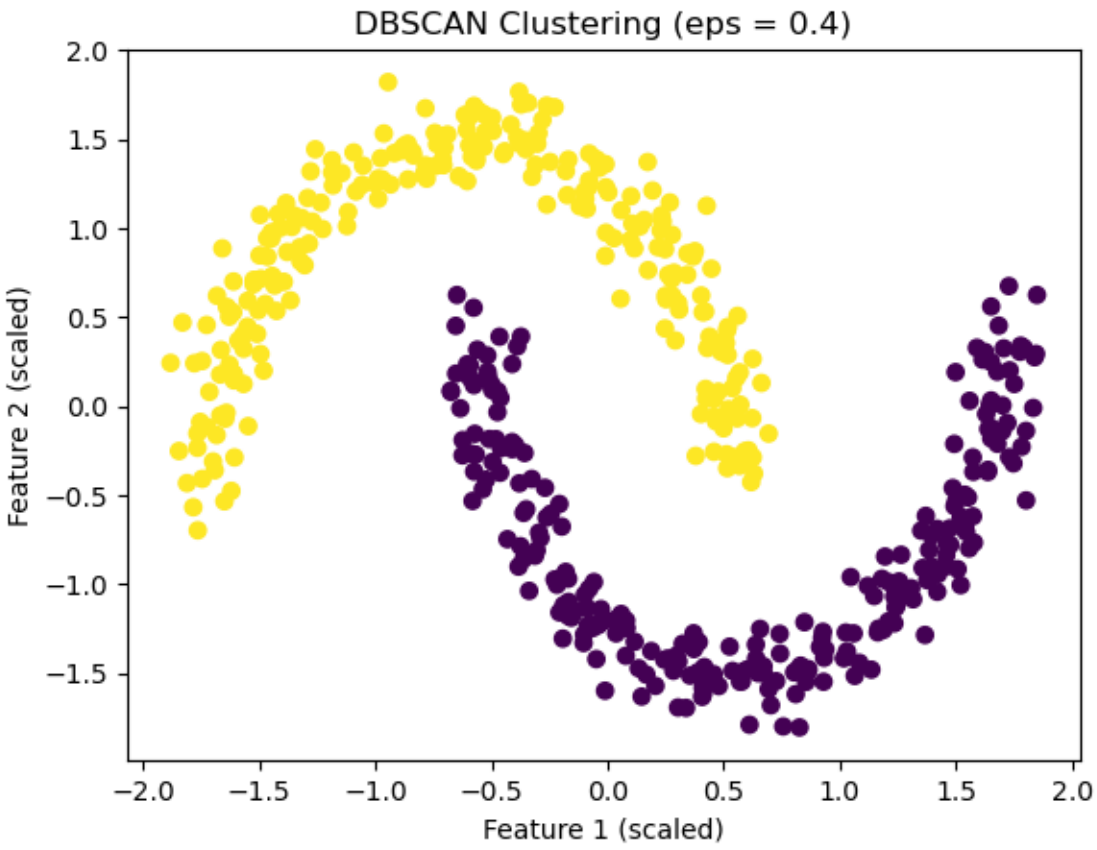

```
eps = 0.2
Clusters found: {np.int64(0), np.int64(1), np.int64(-1)}
```



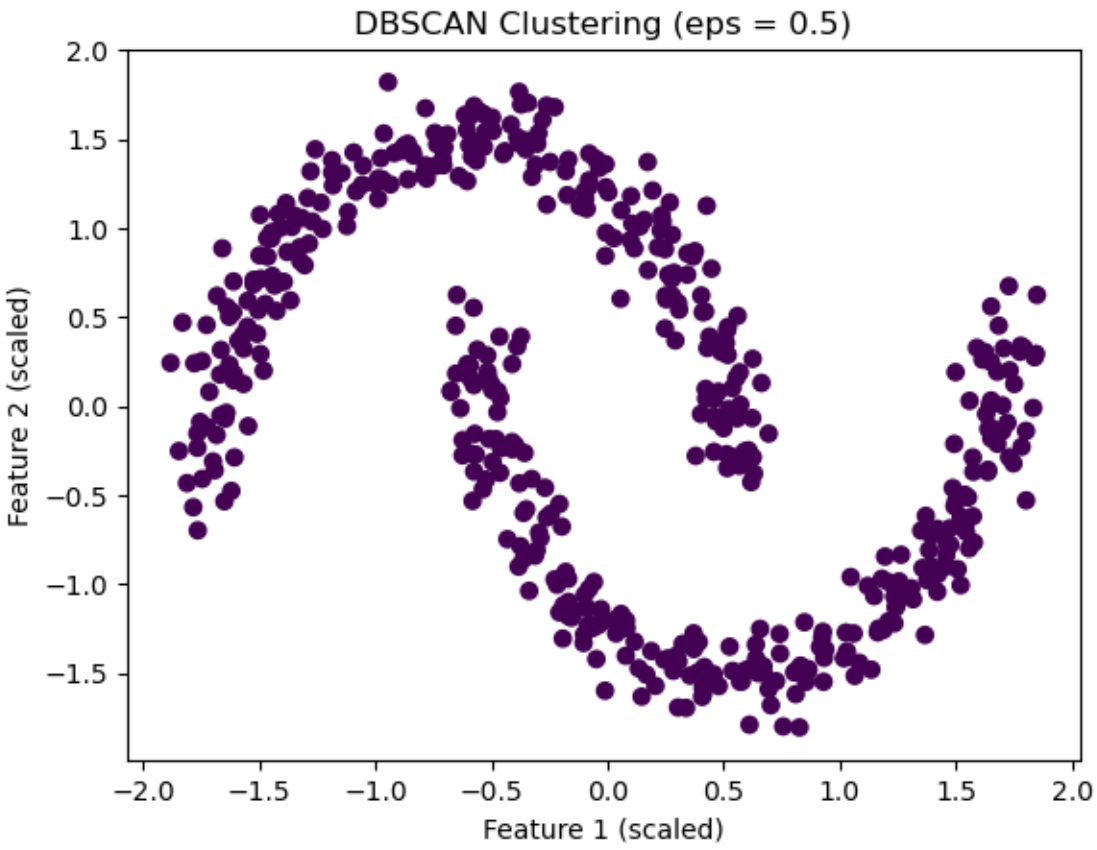

```
eps = 0.3
Clusters found: {np.int64(0), np.int64(1)}
```




```
eps = 0.4
Clusters found: {np.int64(0), np.int64(1)}
```




```
eps = 0.5
Clusters found: {np.int64(0)}
```



[9]:
```
# Cell 7
# Creating a table comparing DBSCAN metrics across different eps and␣
 ↪min_samples values

from sklearn.cluster import DBSCAN
import pandas as pd

eps_values = [0.1, 0.2, 0.3, 0.4, 0.5]
min_samples_values = [3, 5, 7]

results = []

for eps in eps_values:
    for ms in min_samples_values:
        db = DBSCAN(eps=eps, min_samples=ms)
        labels_tmp = db.fit_predict(X_scaled)

        core_samples_mask = np.zeros_like(labels_tmp, dtype=bool)
        core_samples_mask[db.core_sample_indices_] = True
```

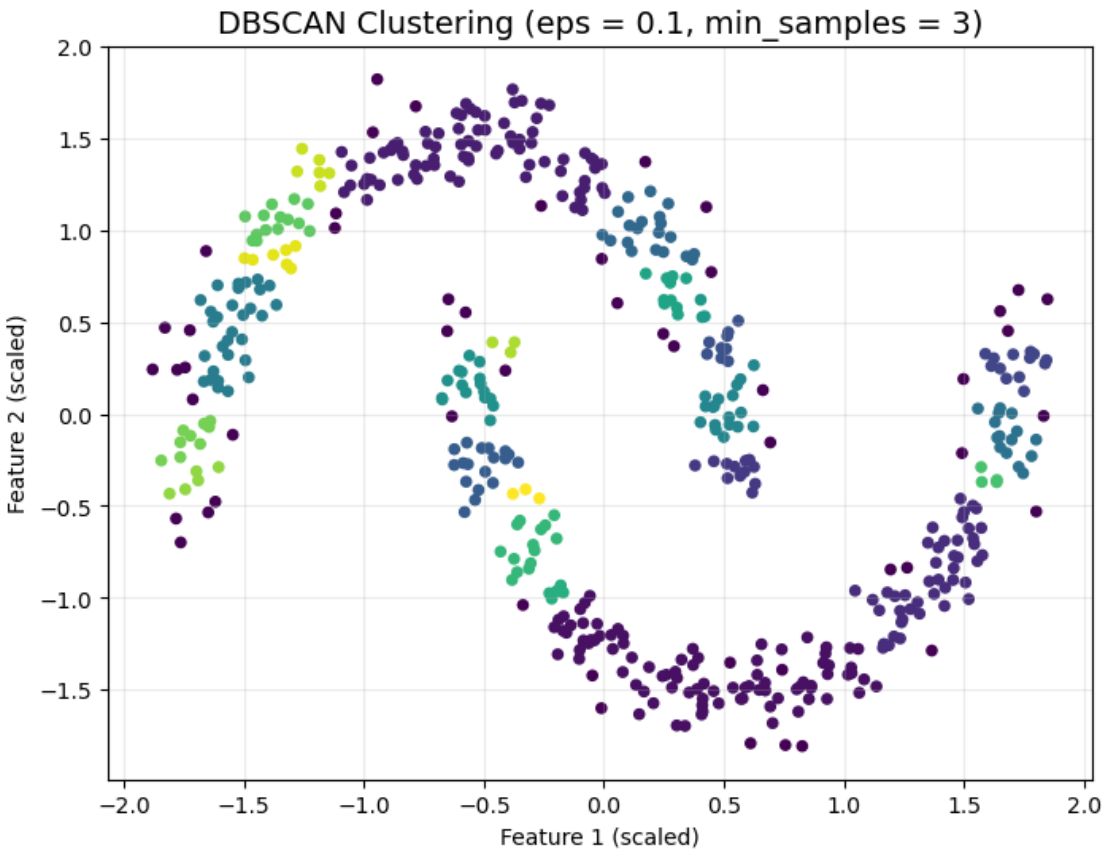
```
        # Count core, border, noise
        core_points = np.sum(core_samples_mask)
        noise_points = np.sum(labels_tmp == -1)
        border_points = len(labels_tmp) - core_points - noise_points

        # Count clusters excluding noise label (-1)
        cluster_labels = set(labels_tmp)
        clusters = len([c for c in cluster_labels if c != -1])

        results.append({
            "eps": eps,
            "min_samples": ms,
            "core_points": core_points,
            "border_points": border_points,
            "noise_points": noise_points,
            "clusters_found": clusters
        })

# Convert to DataFrame
results_df = pd.DataFrame(results)

# Display the table
results_df
```

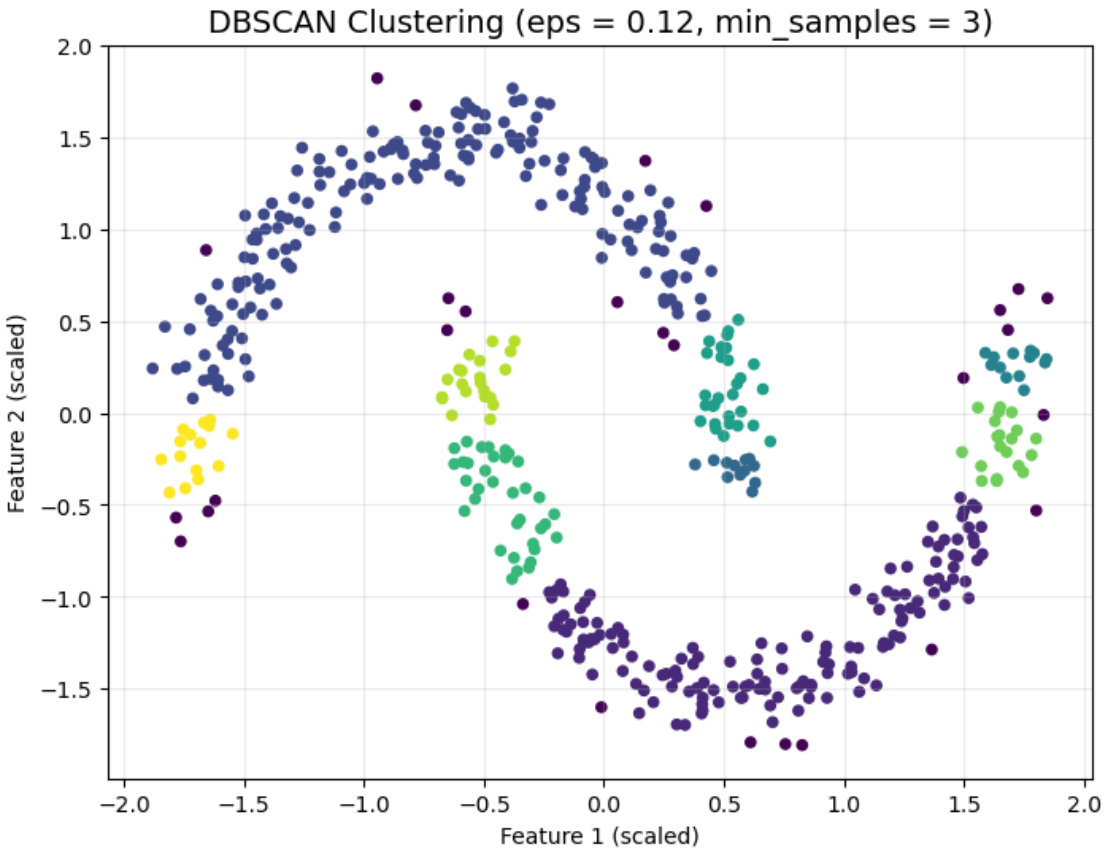
[9]:

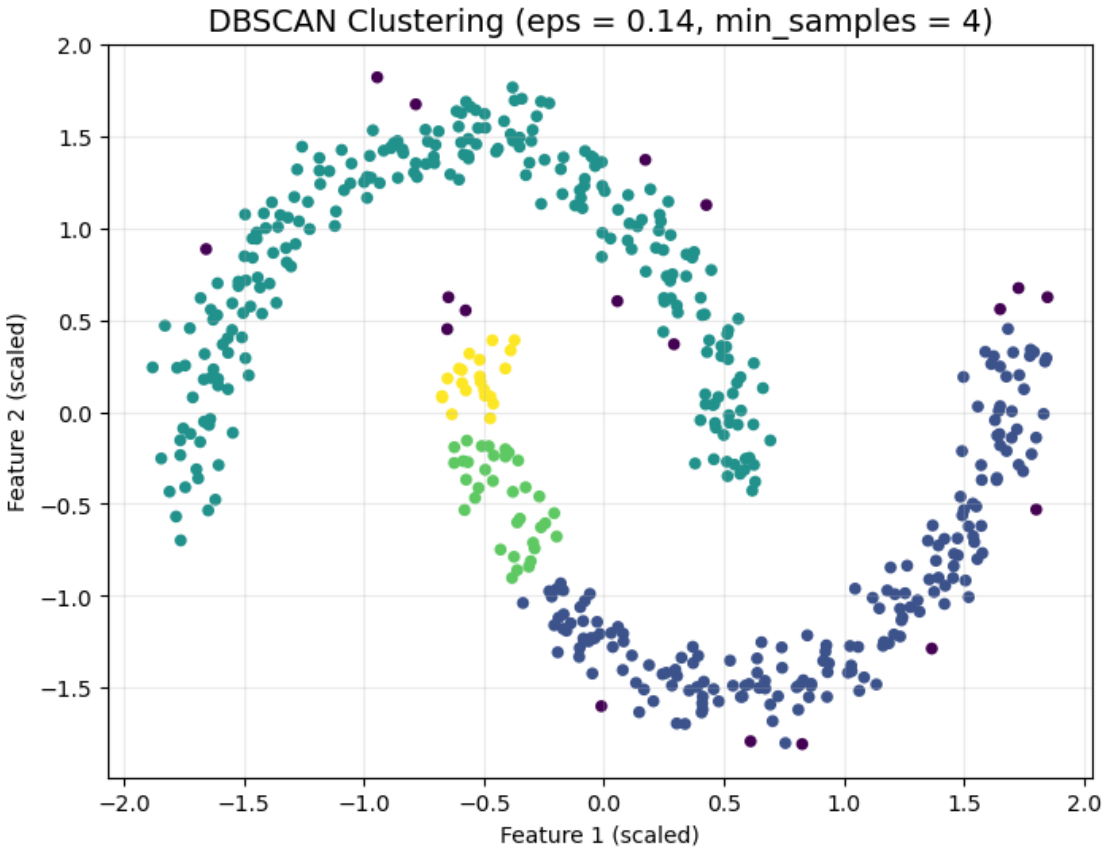
	eps	min_samples	core_points	border_points	noise_points	clusters_found
0	0.1	3	407	42	51	24
1	0.1	5	276	106	118	24
2	0.1	7	103	152	245	21
3	0.2	3	495	3	2	2
4	0.2	5	484	12	4	2
5	0.2	7	472	23	5	2
6	0.3	3	500	0	0	2
7	0.3	5	497	3	0	2
8	0.3	7	495	5	0	2
9	0.4	3	500	0	0	2
10	0.4	5	500	0	0	2
11	0.4	7	500	0	0	2
12	0.5	3	500	0	0	1
13	0.5	5	500	0	0	1
14	0.5	7	500	0	0	1

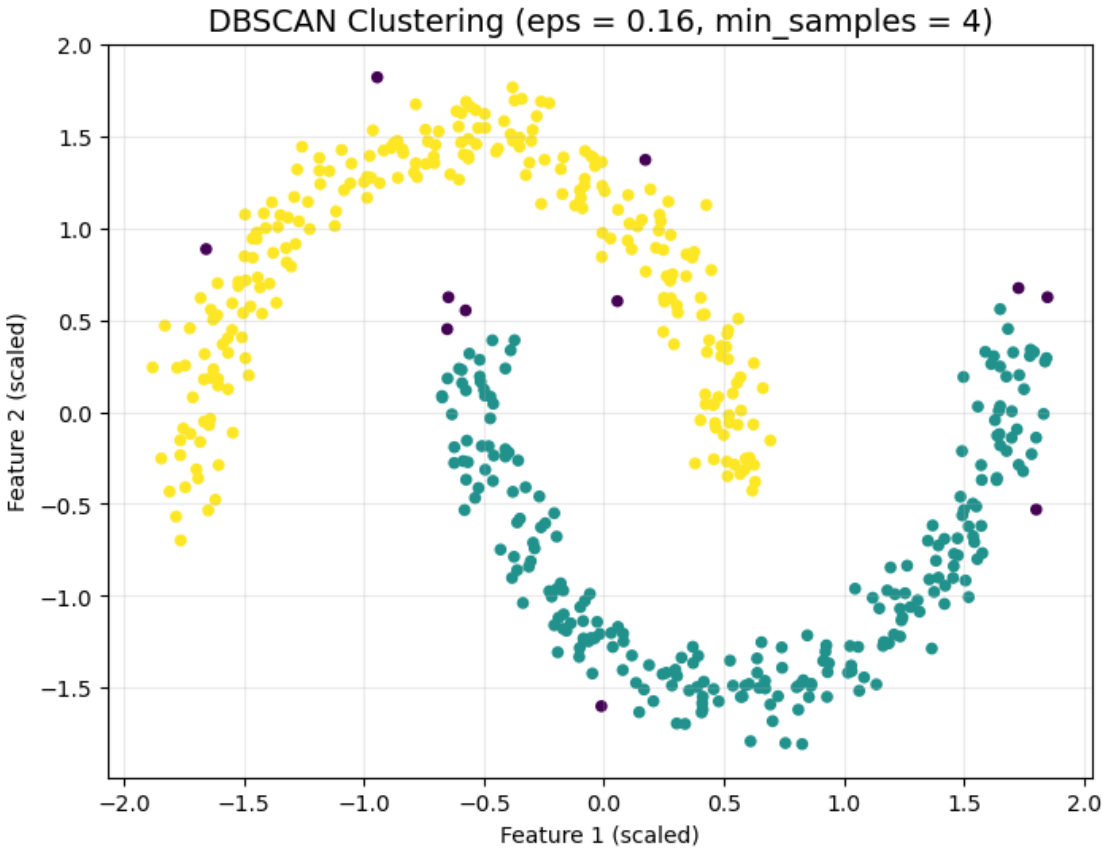
[12]:
```
# Cell: Generate 20 high-quality DBSCAN plots with excellent parameter variety

import matplotlib.pyplot as plt
from sklearn.cluster import DBSCAN

# Carefully chosen parameter combinations (20 total)
```

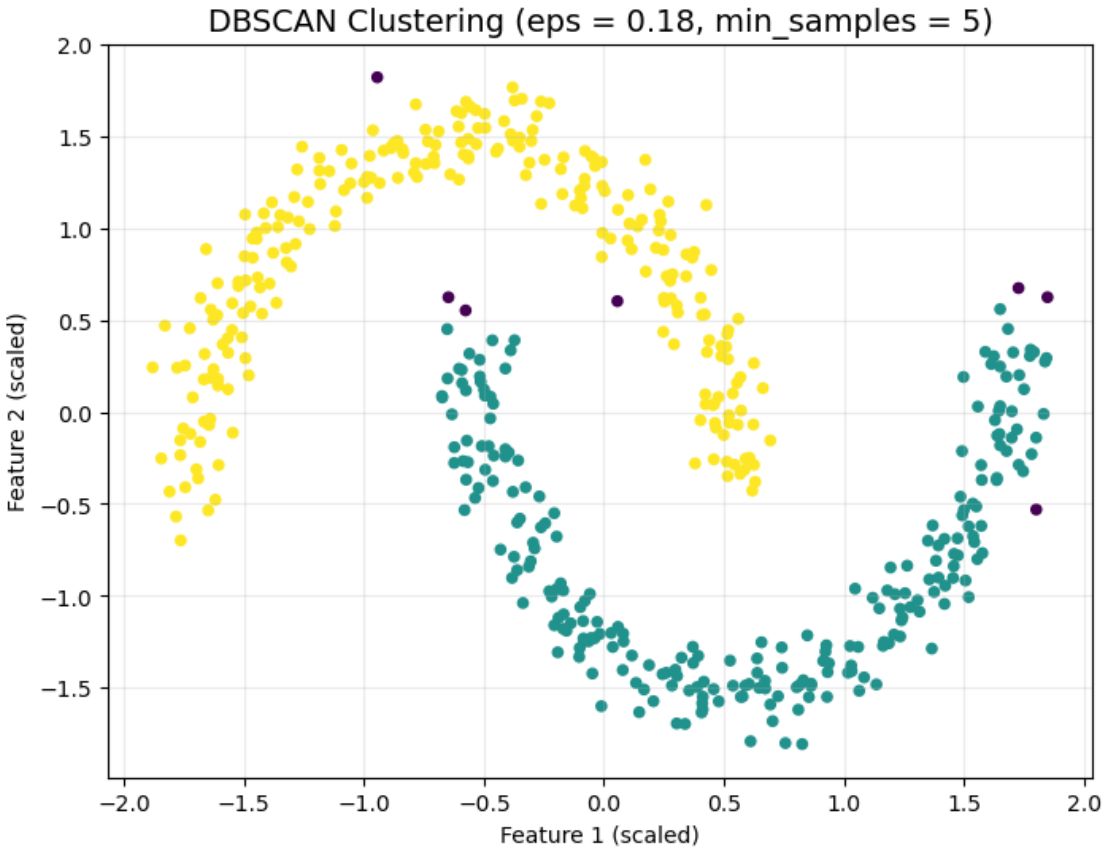
```
params = [
    (0.10, 3), (0.12, 3), (0.14, 4), (0.16, 4), (0.18, 5),
    (0.20, 5), (0.22, 5), (0.24, 6), (0.26, 6), (0.28, 7),
    (0.30, 7), (0.32, 8), (0.34, 8), (0.36, 9), (0.38, 9),
    (0.40, 10), (0.45, 10), (0.50, 12), (0.55, 15), (0.60, 20)
]

for eps, ms in params:
    db = DBSCAN(eps=eps, min_samples=ms)
    labels_tmp = db.fit_predict(X_scaled)

    plt.figure(figsize=(8, 6))
    plt.scatter(
        X_scaled[:, 0], X_scaled[:, 1],
        c=labels_tmp, cmap='viridis', s=20
    )
    plt.title(f"DBSCAN Clustering (eps = {eps}, min_samples = {ms})",␣
 ↪fontsize=14)
    plt.xlabel("Feature 1 (scaled)")
    plt.ylabel("Feature 2 (scaled)")
    plt.grid(True, alpha=0.25)
    plt.show()
```

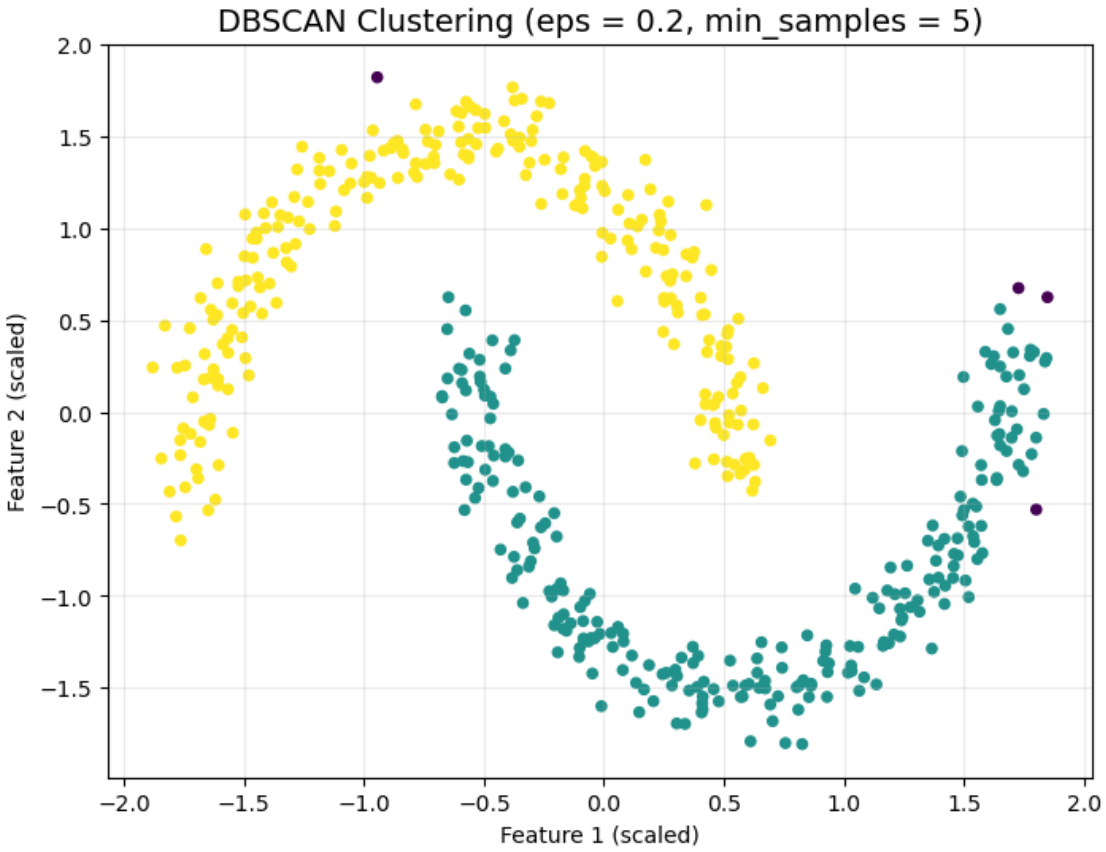


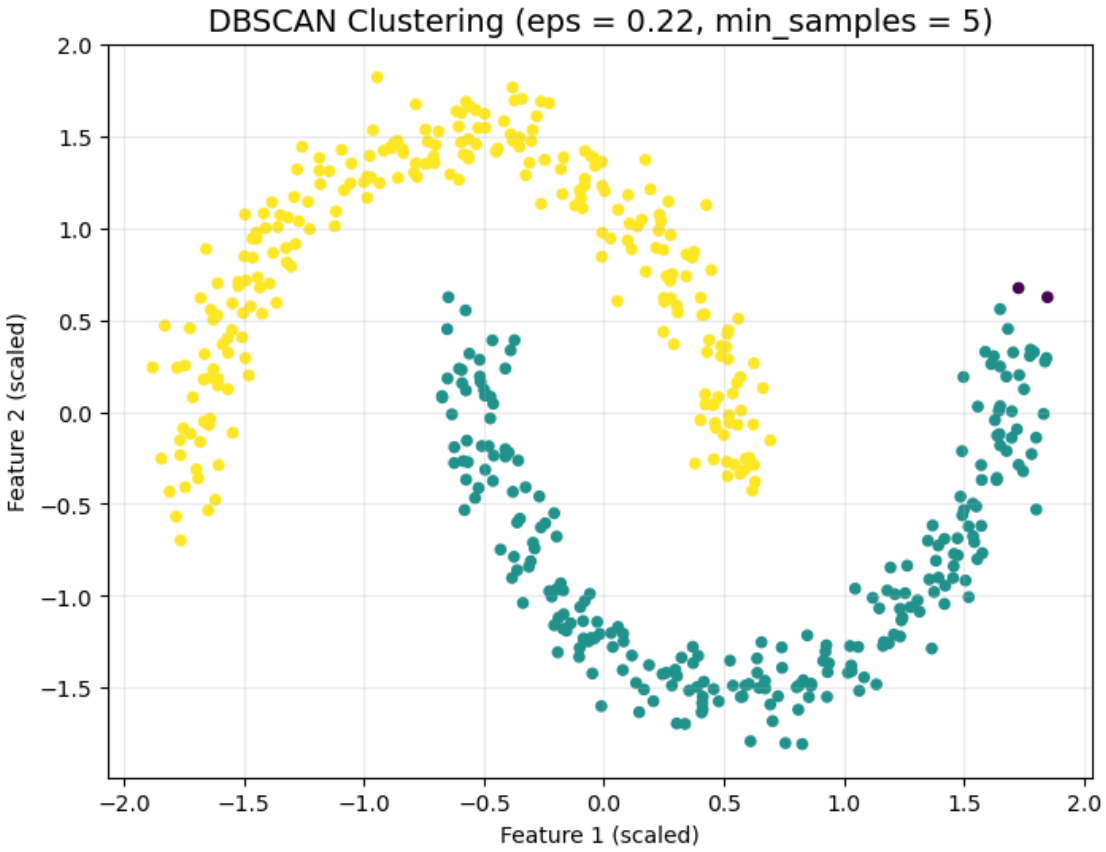

DBSCAN Clustering (eps = 0.12, min_samples = 3)
Feature 2 (scaled)
Feature 1 (scaled)
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

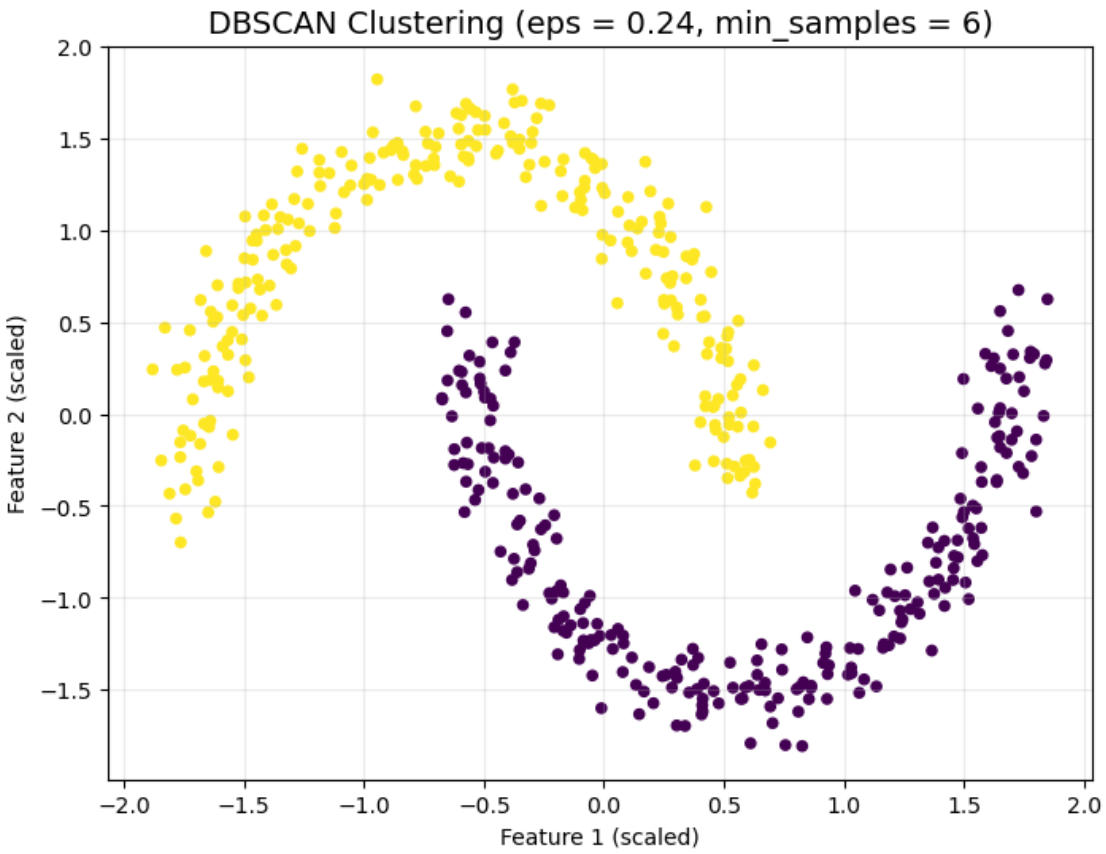

DBSCAN Clustering (eps = 0.14, min_samples = 4)
Feature 2 (scaled)
Feature 1 (scaled)
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

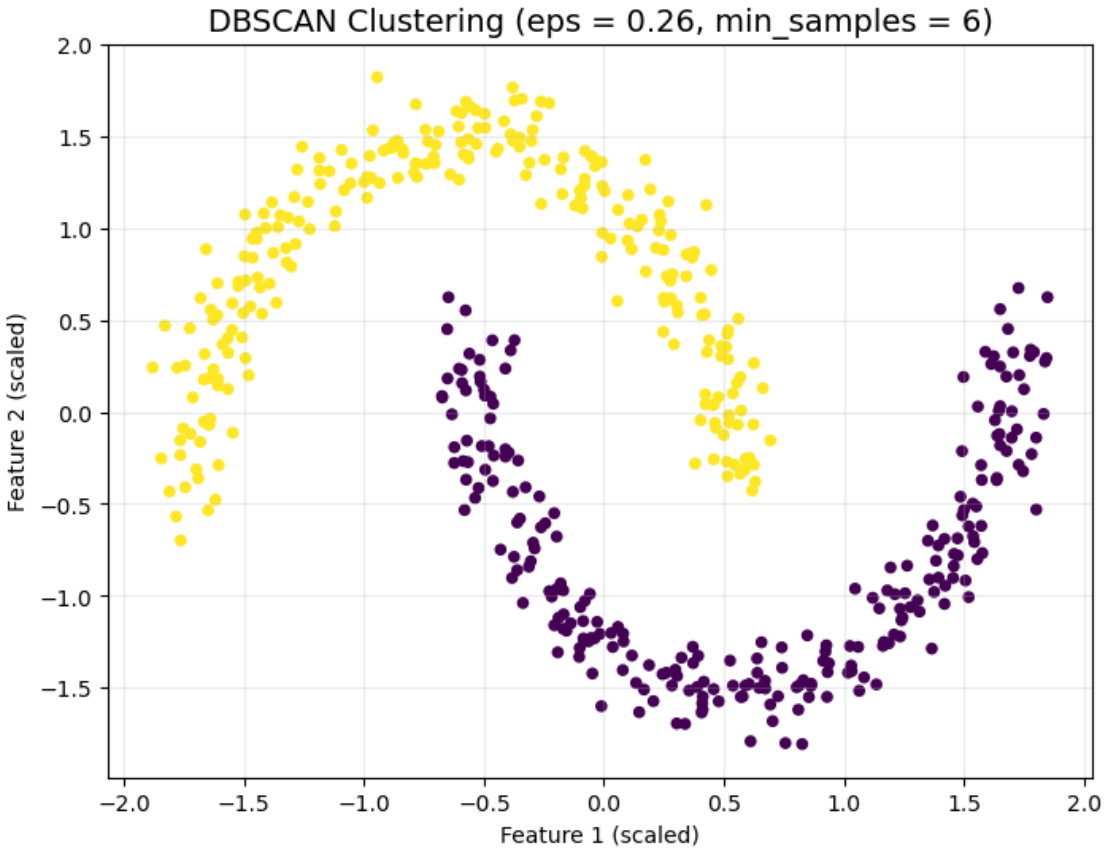

DBSCAN Clustering (eps = 0.16, min_samples = 4)
Feature 2 (scaled)
Feature 1 (scaled)
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

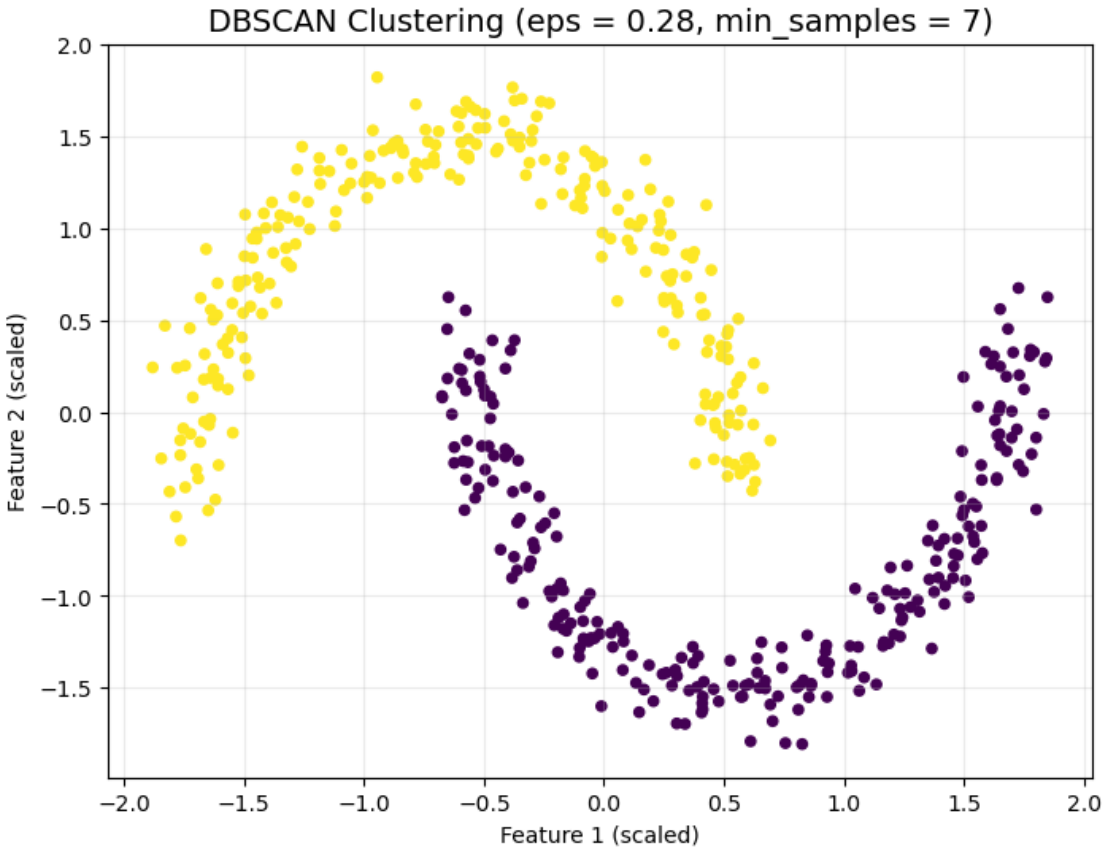

DBSCAN Clustering (eps = 0.18, min_samples = 5)
Feature 2 (scaled)
Feature 1 (scaled)
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

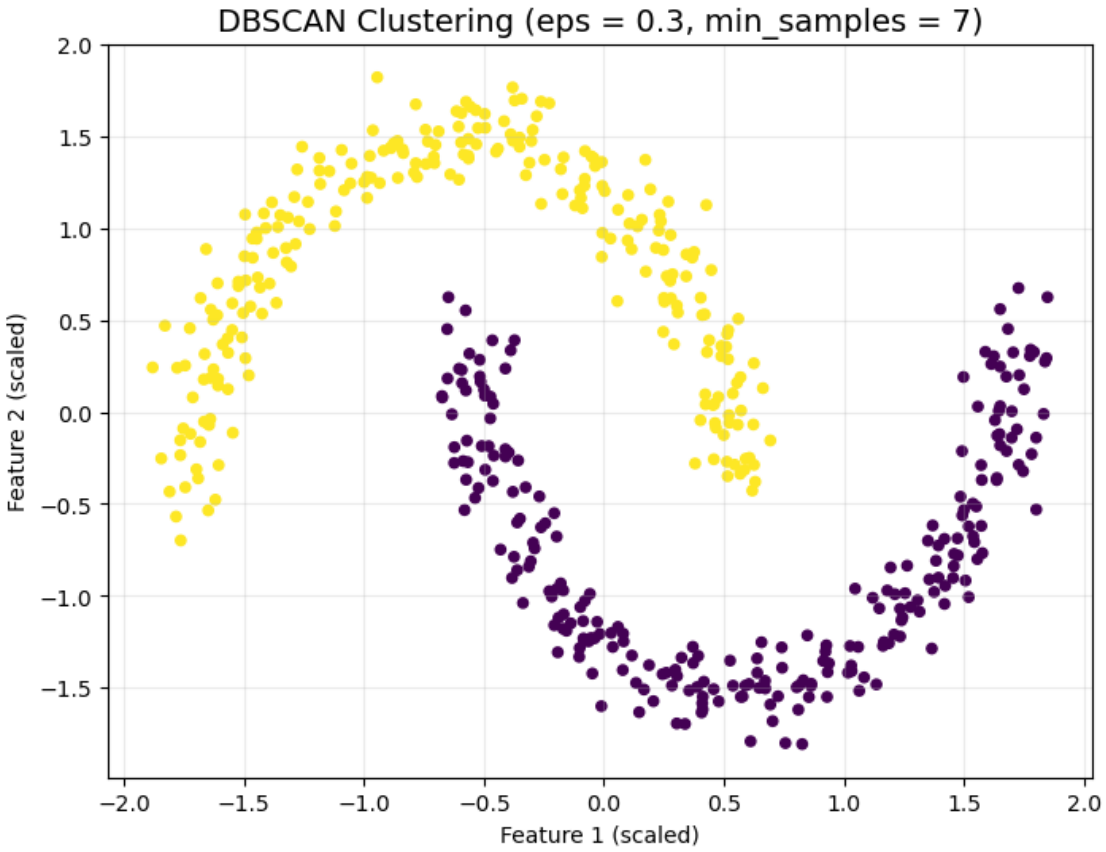

DBSCAN Clustering (eps = 0.2, min_samples = 5)
Feature 2 (scaled)
Feature 1 (scaled)
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

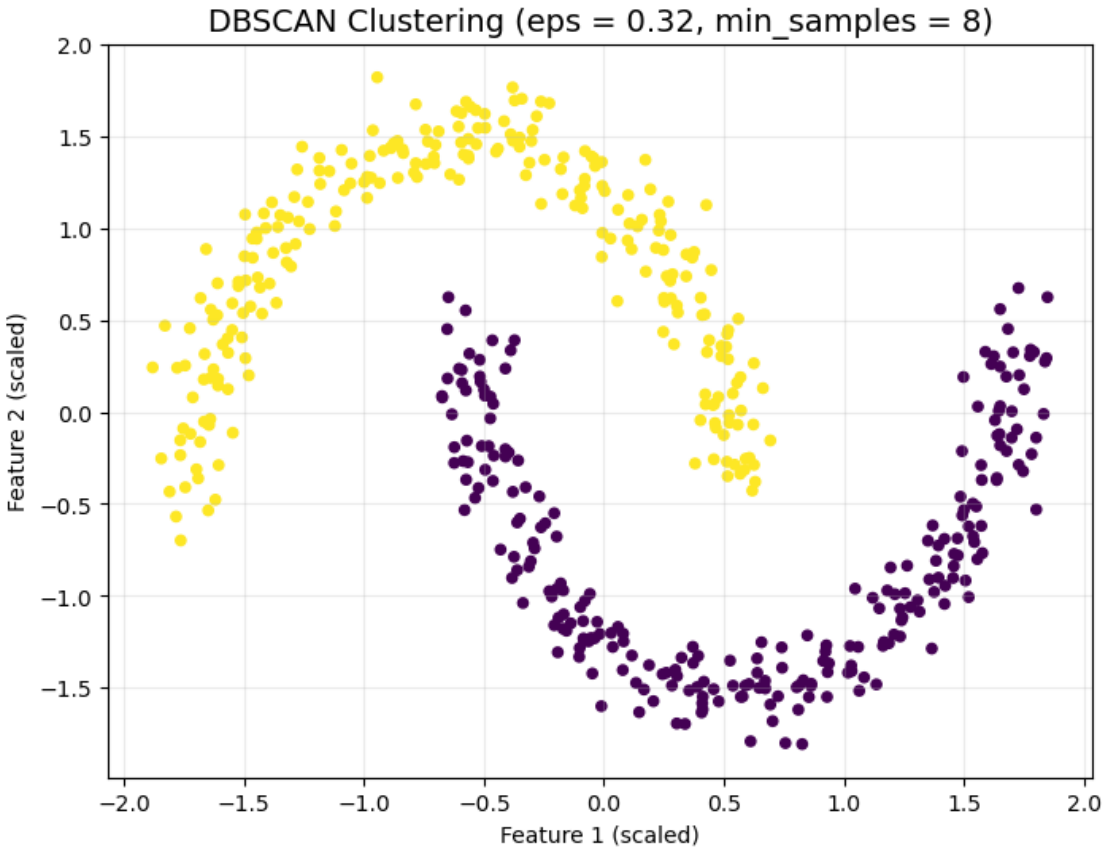

DBSCAN Clustering (eps = 0.22, min_samples = 5)
Feature 2 (scaled)
Feature 1 (scaled)
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

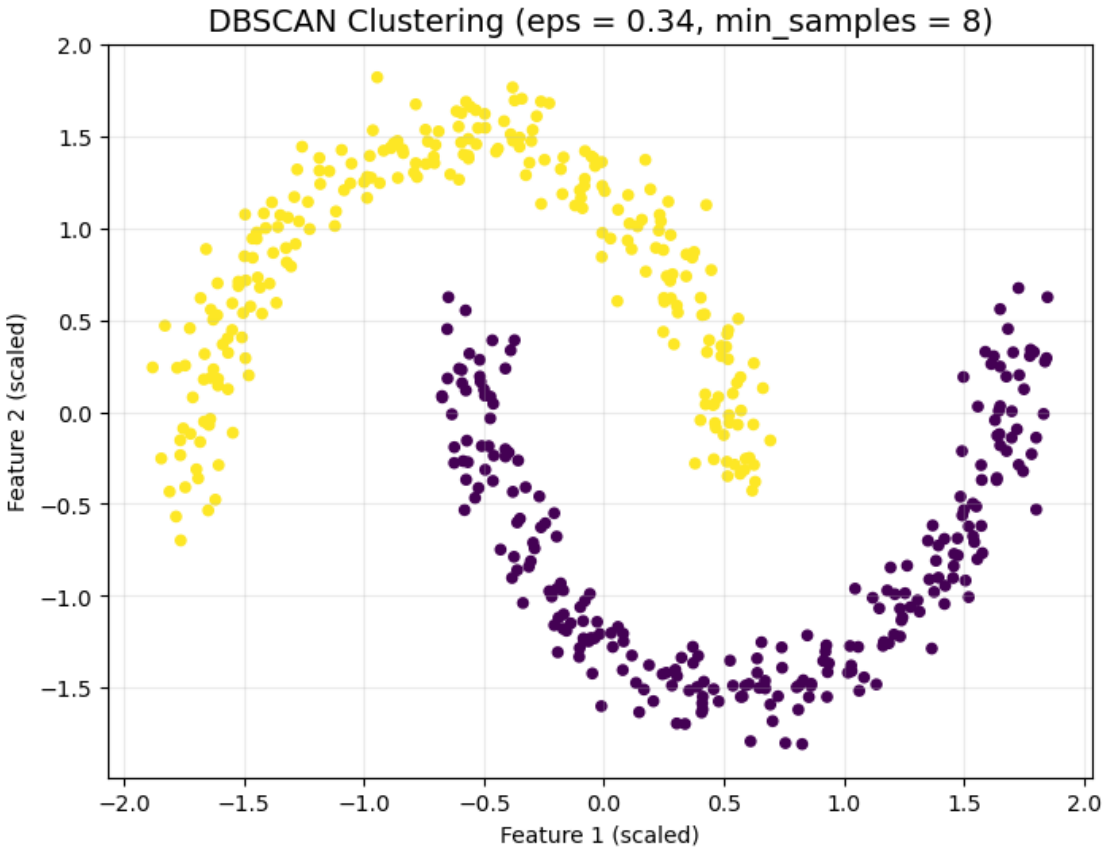

DBSCAN Clustering (eps = 0.24, min_samples = 6)
Feature 2 (scaled)
Feature 1 (scaled)
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

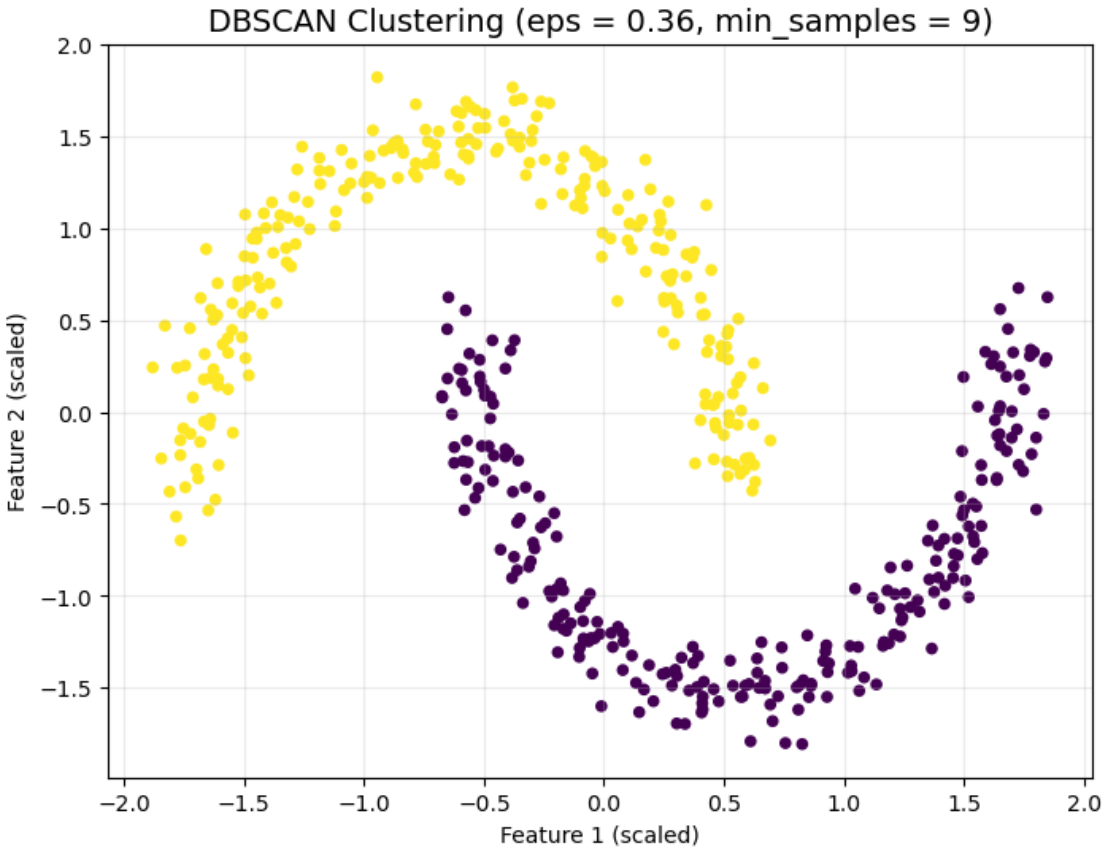

DBSCAN Clustering (eps = 0.26, min_samples = 6)
Feature 2 (scaled)
Feature 1 (scaled)
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

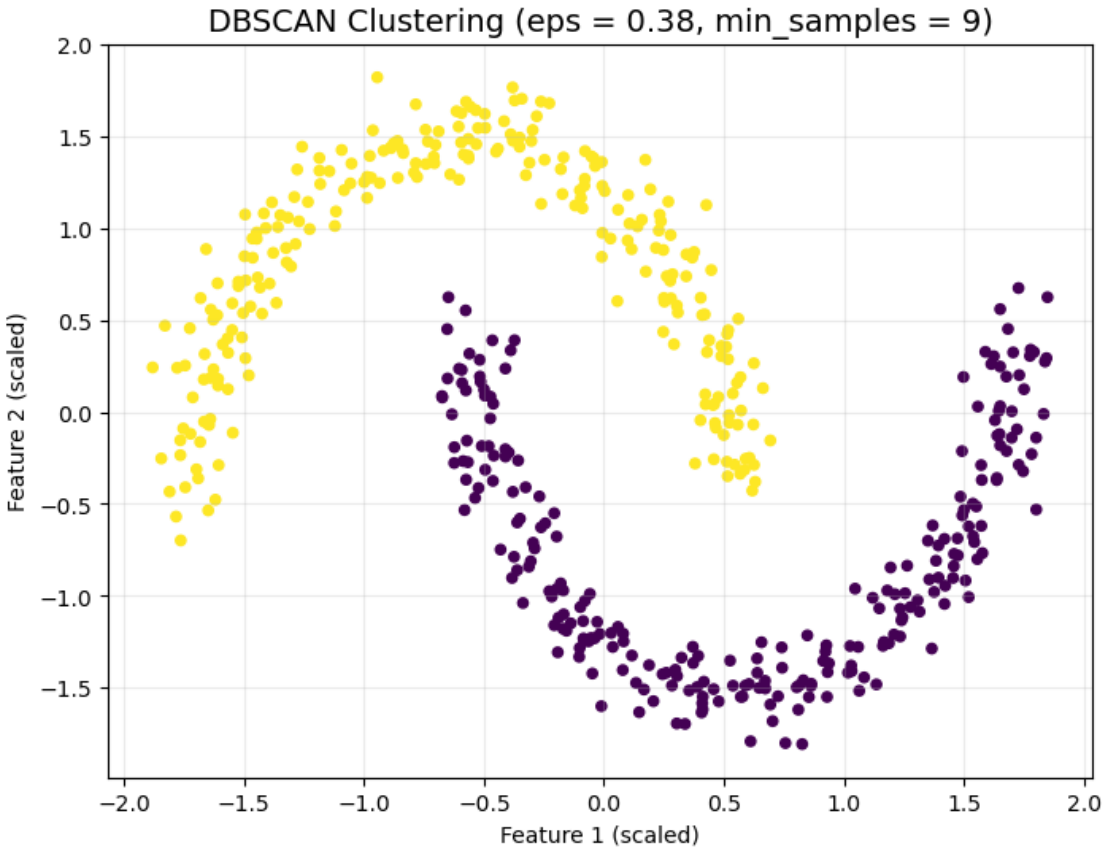

DBSCAN Clustering (eps = 0.28, min_samples = 7)
Feature 2 (scaled)
Feature 1 (scaled)
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

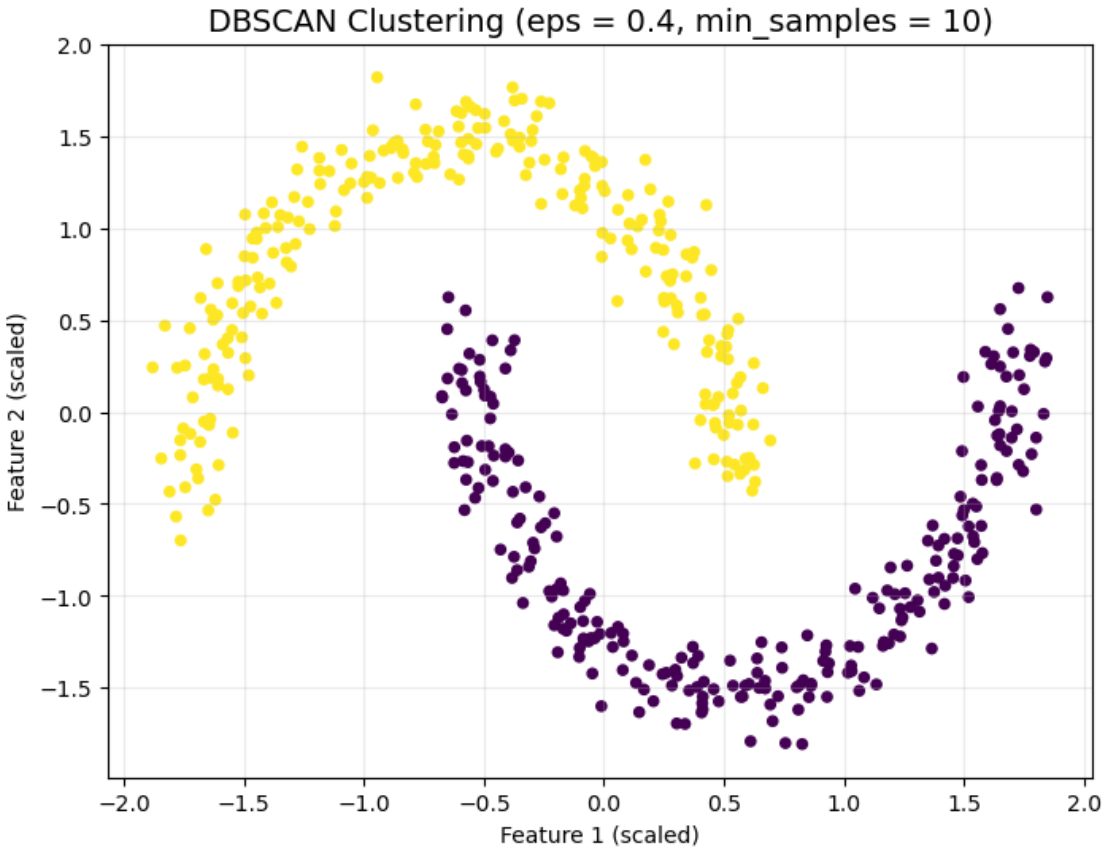

DBSCAN Clustering (eps = 0.3, min_samples = 7)
Feature 2 (scaled)
Feature 1 (scaled)
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

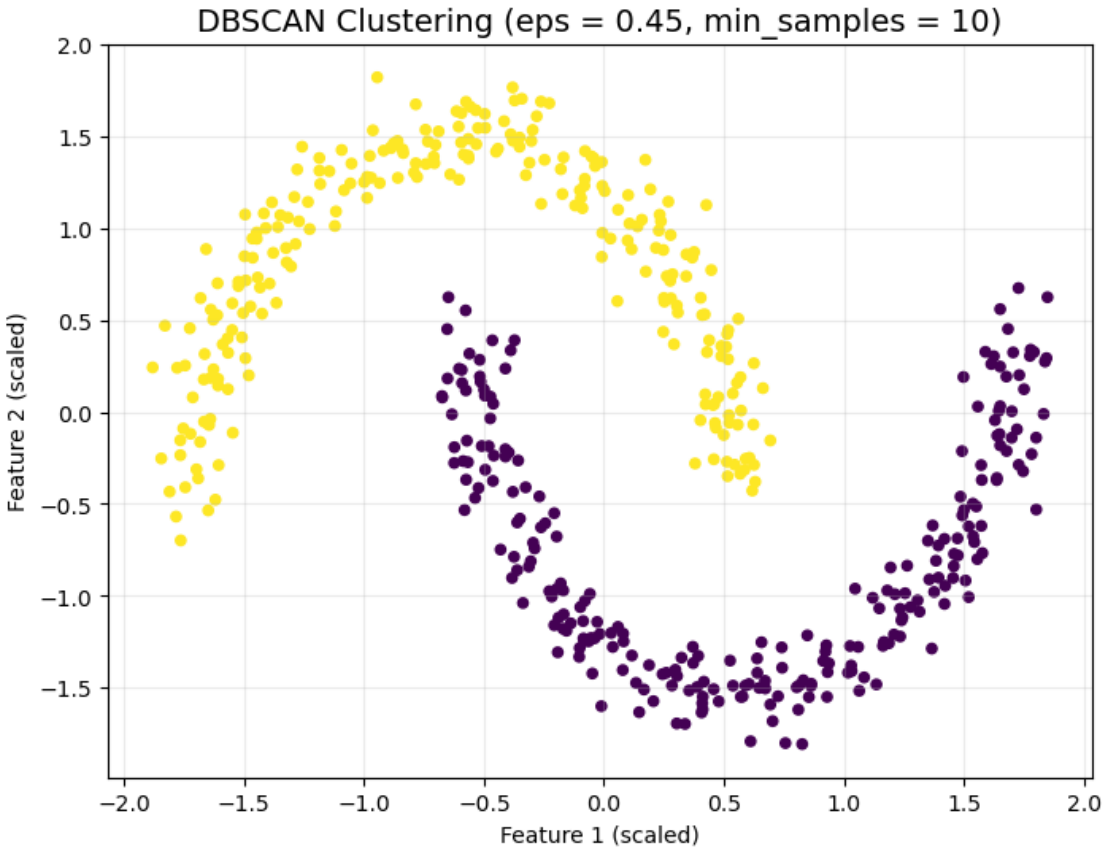

DBSCAN Clustering (eps = 0.32, min_samples = 8)
Feature 2 (scaled)
Feature 1 (scaled)
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

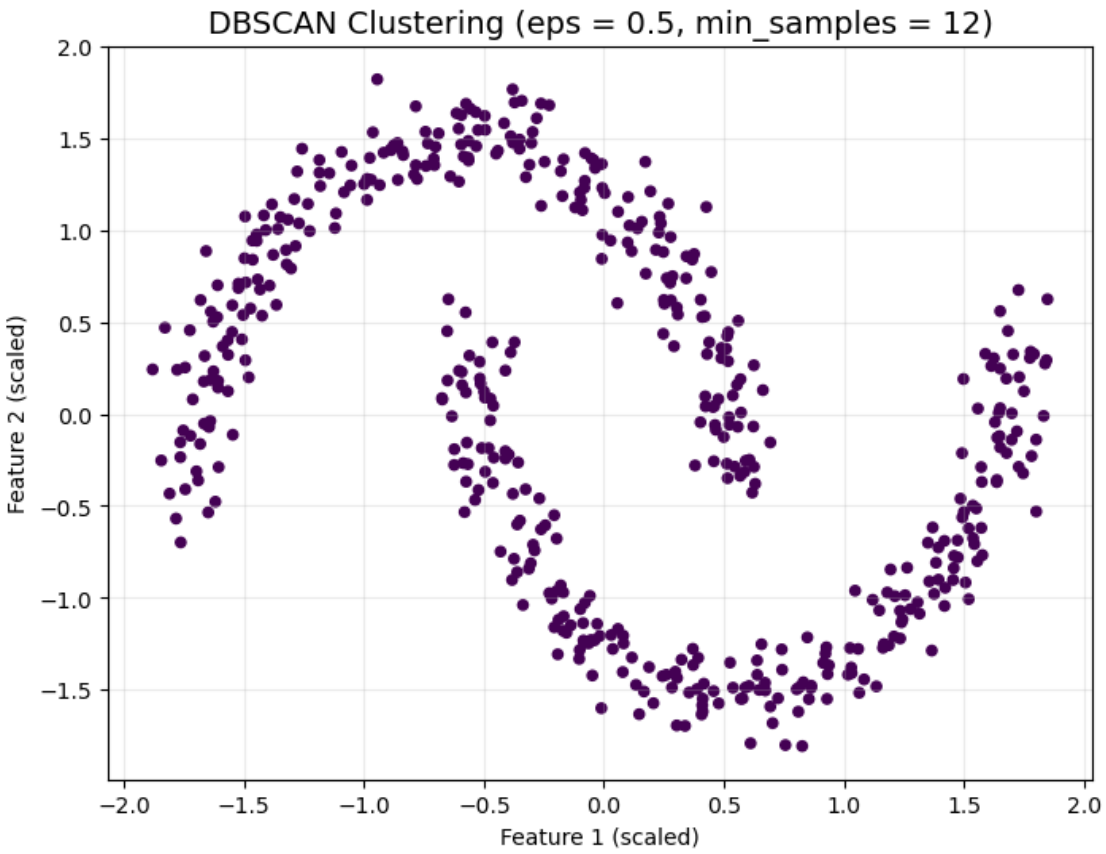

DBSCAN Clustering (eps = 0.34, min_samples = 8)
Feature 2 (scaled)
Feature 1 (scaled)
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

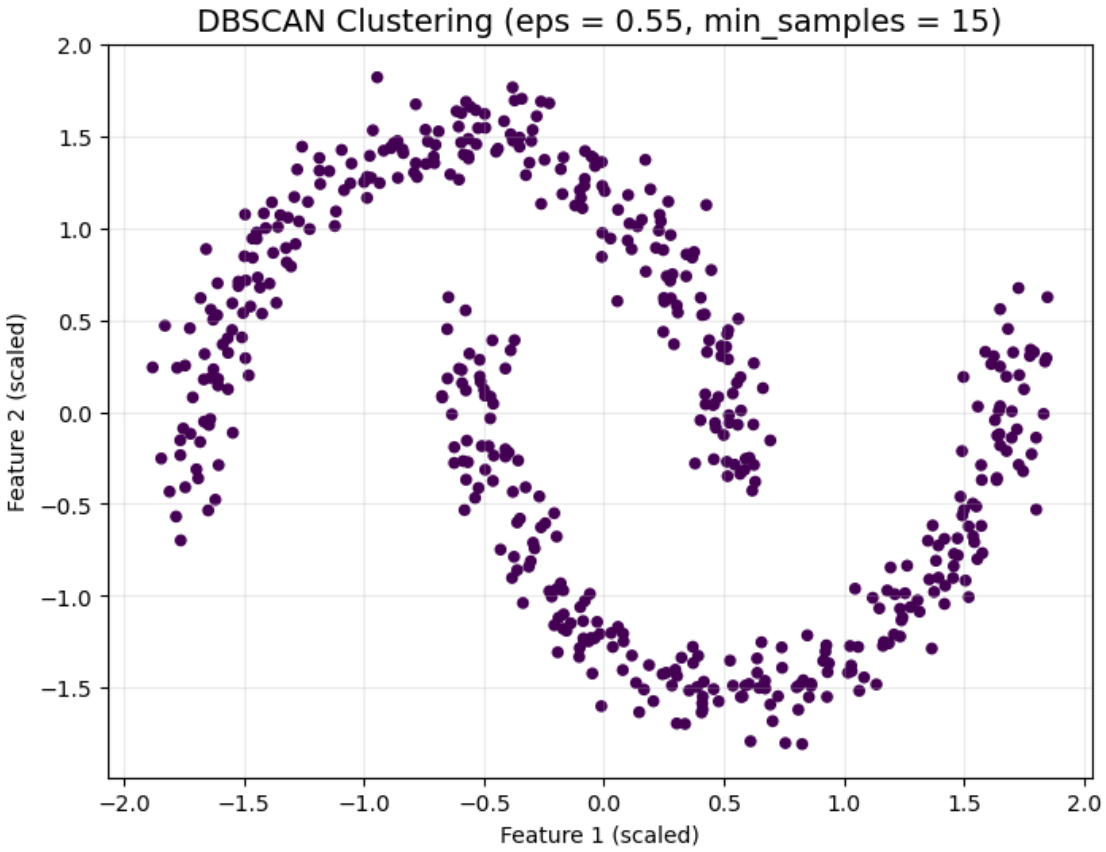

DBSCAN Clustering (eps = 0.36, min_samples = 9)
Feature 2 (scaled)
Feature 1 (scaled)
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

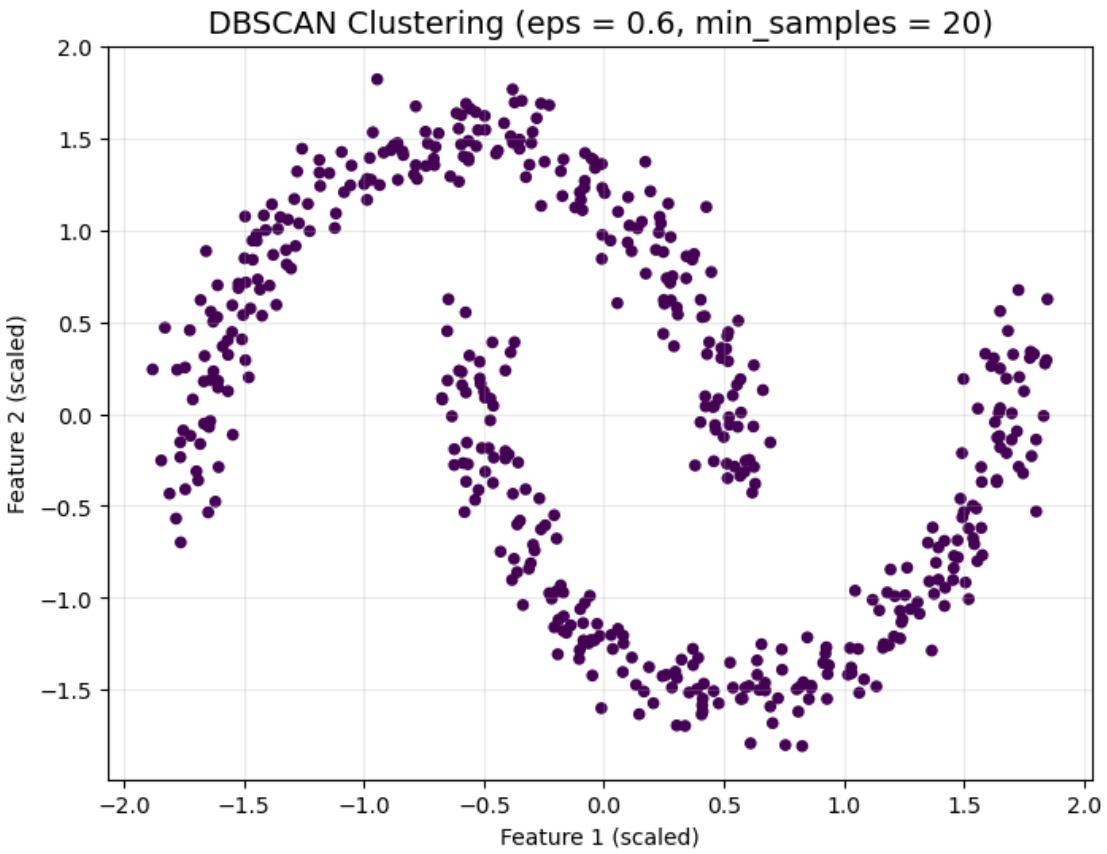

DBSCAN Clustering (eps = 0.38, min_samples = 9)
Feature 2 (scaled)
Feature 1 (scaled)
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0


DBSCAN Clustering (eps = 0.4, min_samples = 10)
Feature 2 (scaled)
Feature 1 (scaled)
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0


DBSCAN Clustering (eps = 0.45, min_samples = 10)
Feature 2 (scaled)
Feature 1 (scaled)
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0


DBSCAN Clustering (eps = 0.5, min_samples = 12)
Feature 2 (scaled)
Feature 1 (scaled)
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0


DBSCAN Clustering (eps = 0.55, min_samples = 15)
Feature 2 (scaled)
Feature 1 (scaled)
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0



[13]:
```
# Cell: Silhouette Score table for all 20 parameter combinations

import pandas as pd
from sklearn.metrics import silhouette_score

sil_results = []

for eps, ms in params:
    db = DBSCAN(eps=eps, min_samples=ms)
    labels_tmp = db.fit_predict(X_scaled)

    unique_labels = set(labels_tmp)
    valid_clusters = [c for c in unique_labels if c != -1]

    # Compute silhouette only if at least 2 valid clusters
    if len(valid_clusters) >= 2:
        score = silhouette_score(X_scaled, labels_tmp)
    else:
        score = None  # cannot compute
```

```
    sil_results.append({
        "eps": eps,
        "min_samples": ms,
        "clusters_found": len(valid_clusters),
        "noise_points": list(labels_tmp).count(-1),
        "silhouette_score": score
    })

sil_df = pd.DataFrame(sil_results)
sil_df
```

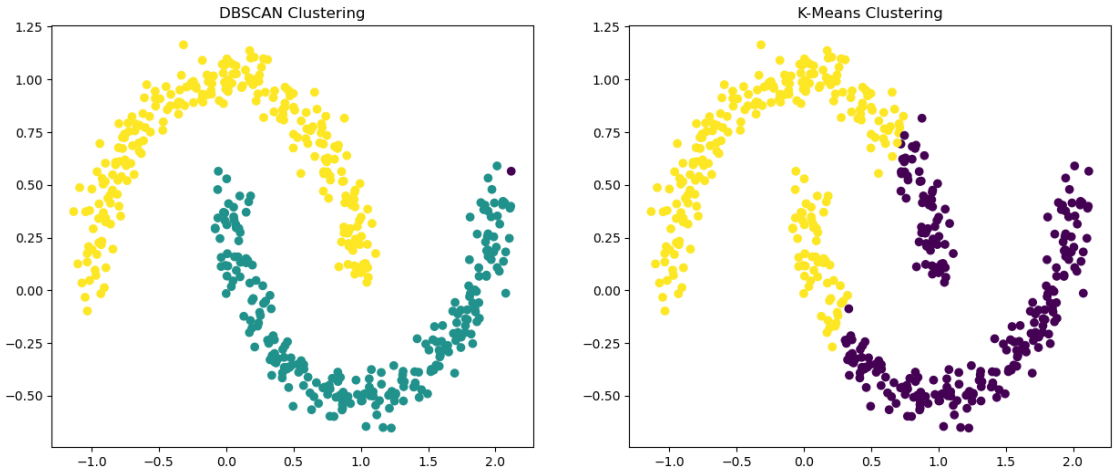
[13]:

	eps	min_samples	clusters_found	noise_points	silhouette_score
0	0.10	3	24	51	0.162947
1	0.12	3	9	28	0.185865
2	0.14	4	4	18	0.108356
3	0.16	4	2	11	0.258897
4	0.18	5	2	7	0.257724
5	0.20	5	2	4	0.296218
6	0.22	5	2	2	0.263170
7	0.24	6	2	0	0.385733
8	0.26	6	2	0	0.385733
9	0.28	7	2	0	0.385733
10	0.30	7	2	0	0.385733
11	0.32	8	2	0	0.385733
12	0.34	8	2	0	0.385733
13	0.36	9	2	0	0.385733
14	0.38	9	2	0	0.385733
15	0.40	10	2	0	0.385733
16	0.45	10	2	0	0.385733
17	0.50	12	1	0	NaN
18	0.55	15	1	0	NaN
19	0.60	20	1	0	NaN

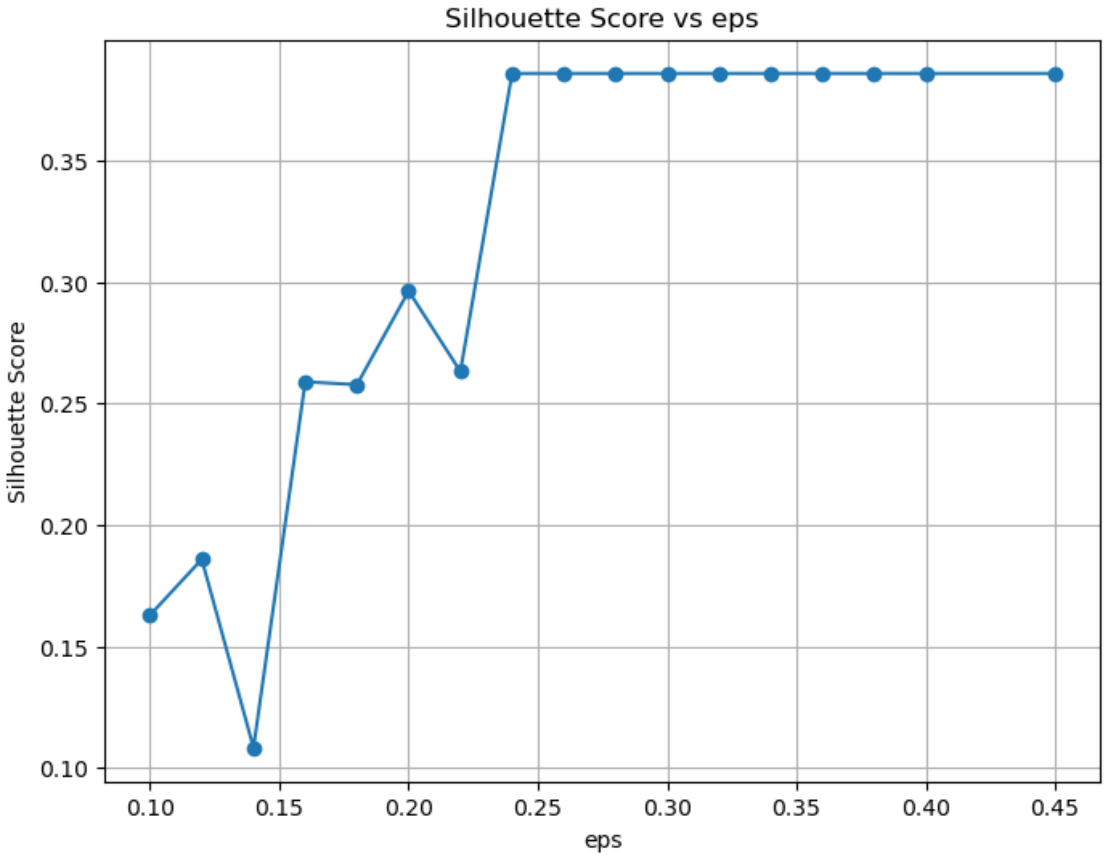
[14]:
```
from sklearn.cluster import KMeans

# DBSCAN clustering
dbscan = DBSCAN(eps=0.15, min_samples=5)
dbscan_labels = dbscan.fit_predict(X)

# K-Means clustering
kmeans = KMeans(n_clusters=2, random_state=42)
kmeans_labels = kmeans.fit_predict(X)

# Visualize the results
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 6))

ax1.scatter(X[:, 0], X[:, 1], c=dbscan_labels, cmap='viridis')
```

```
ax1.set_title('DBSCAN Clustering')

ax2.scatter(X[:, 0], X[:, 1], c=kmeans_labels, cmap='viridis')
ax2.set_title('K-Means Clustering')

plt.show()
```

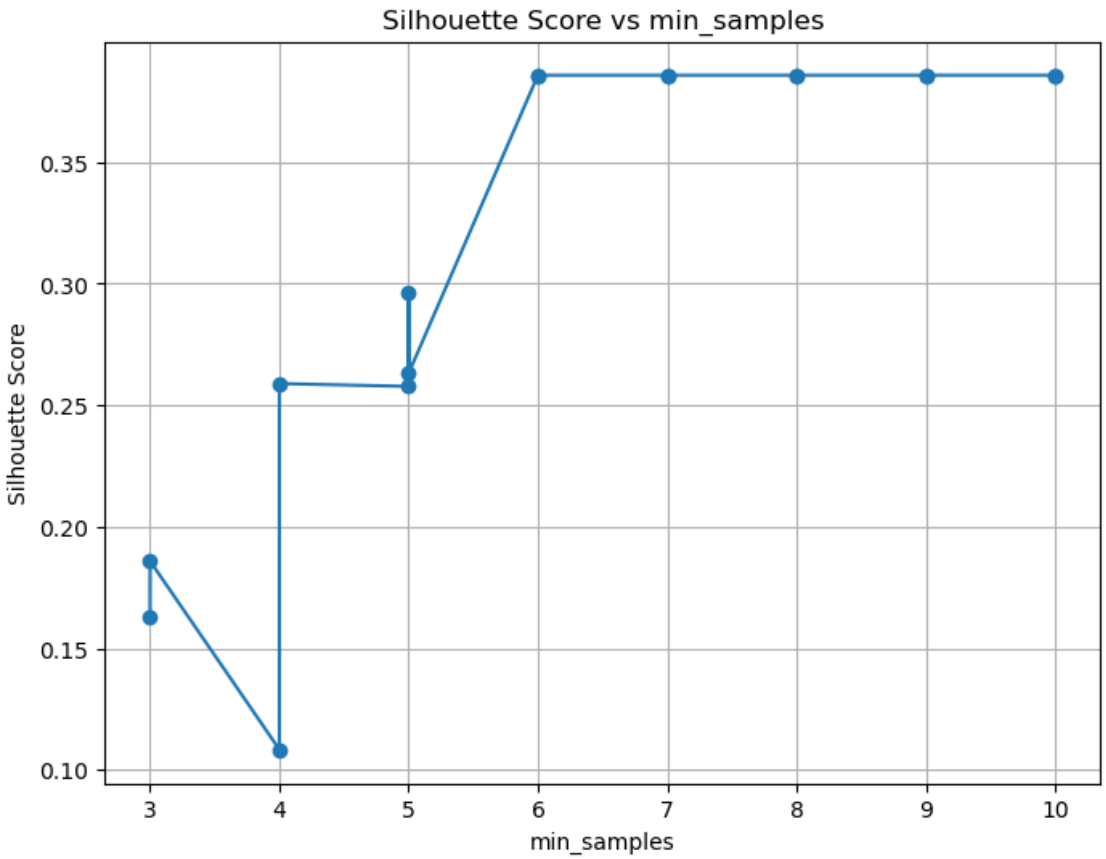
```
C:\Users\ASUS\Downloads\DataMining\Lib\site-
packages\sklearn\cluster\_kmeans.py:1419: UserWarning: KMeans is known to have a
memory leak on Windows with MKL, when there are less chunks than available
threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=2.
  warnings.warn(
```



[16]:
```
# Cell: Plot Silhouette Score vs eps

# Keep only rows where silhouette score is valid (not None)
valid_df = sil_df.dropna(subset=["silhouette_score"])

plt.figure(figsize=(8, 6))
plt.plot(
    valid_df["eps"],
    valid_df["silhouette_score"],
    marker='o'
)
plt.xlabel("eps")
plt.ylabel("Silhouette Score")
plt.title("Silhouette Score vs eps")
plt.grid(True)
plt.show()
```

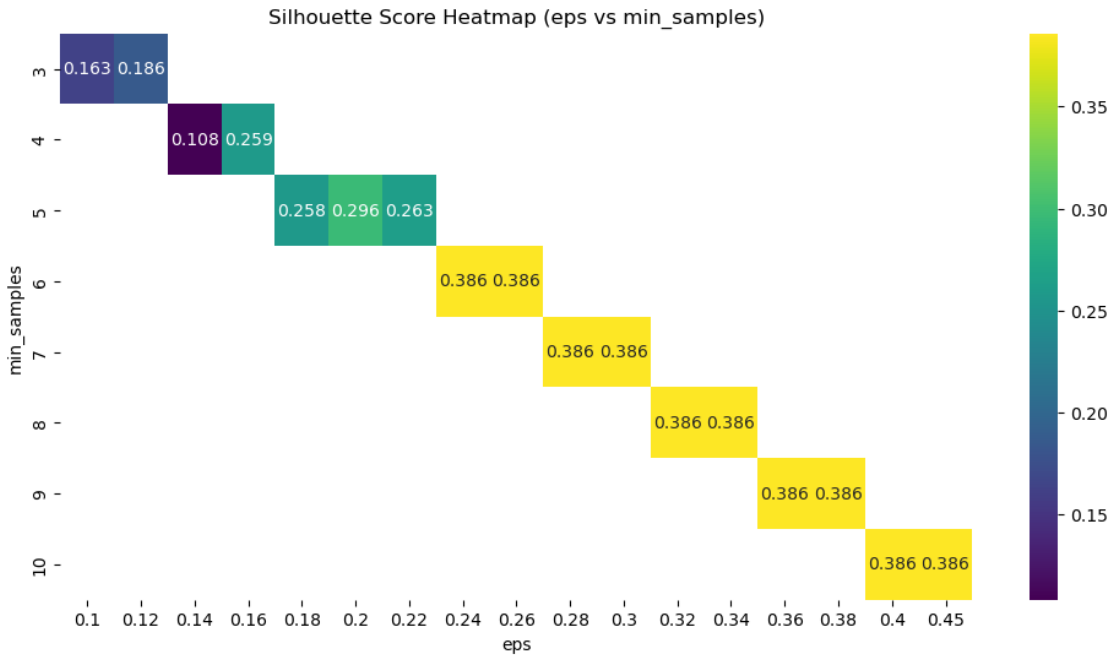


[17]:
```
# Cell: Plot Silhouette Score vs min_samples

# Keep only valid silhouette rows
valid_df = sil_df.dropna(subset=["silhouette_score"])

plt.figure(figsize=(8, 6))
plt.plot(
    valid_df["min_samples"],
    valid_df["silhouette_score"],
    marker='o'
)
plt.xlabel("min_samples")
plt.ylabel("Silhouette Score")
plt.title("Silhouette Score vs min_samples")
plt.grid(True)
plt.show()
```



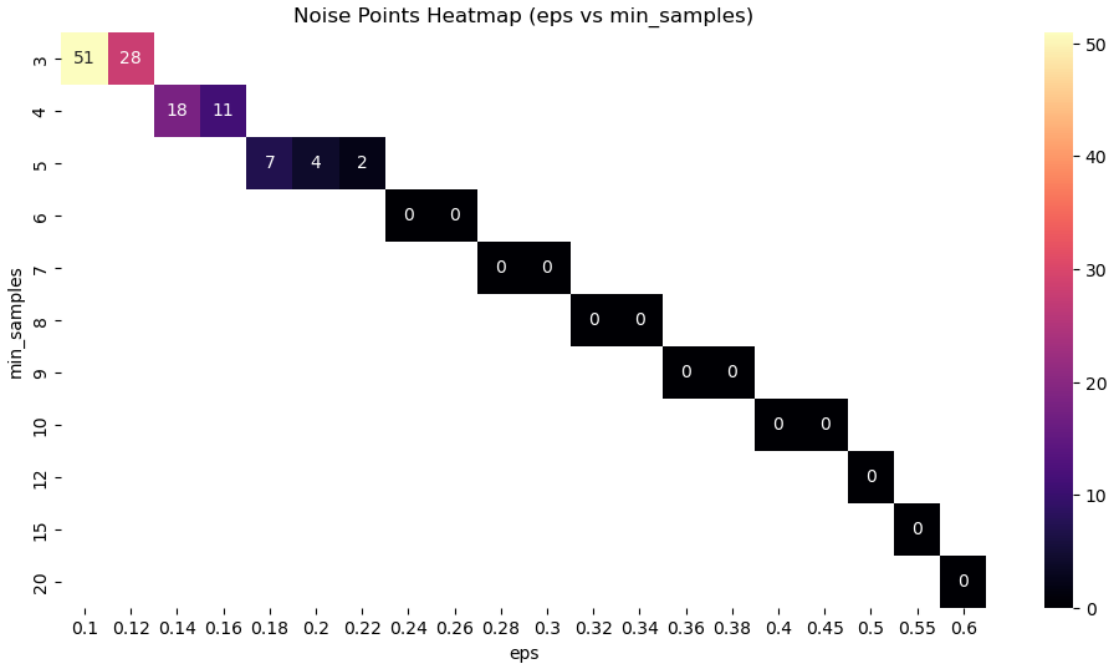
[18]:
```
# Cell: Heatmap of Silhouette Score for eps × min_samples

import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# Filter out invalid silhouette scores
valid_df = sil_df.dropna(subset=["silhouette_score"])

# Pivot for heatmap
heatmap_df = valid_df.pivot(index="min_samples", columns="eps",␣
 ↪values="silhouette_score")

plt.figure(figsize=(12, 6))
sns.heatmap(heatmap_df, annot=True, fmt=".3f", cmap="viridis")
plt.title("Silhouette Score Heatmap (eps vs min_samples)")
plt.xlabel("eps")
plt.ylabel("min_samples")
```

```
plt.show()
```



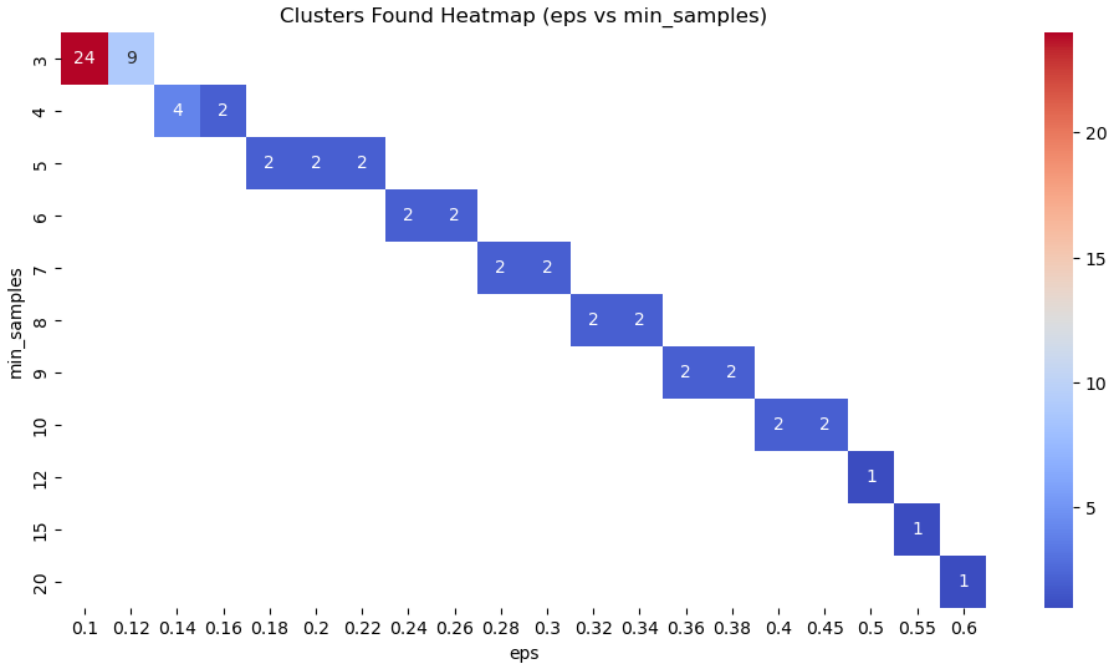
1. The silhouette score is **low at very small eps values**, because DBSCAN breaks the moons into many tiny clusters or noise.
2. As eps increases to the **0.20–0.30 range**, the silhouette score becomes highest (~0.38), indicating **ideal cluster separation**.
3. Increasing min_samples also stabilizes clustering, giving consistent high silhouette scores once eps is in the optimal zone.
4. Very large eps values still give high silhouette scores, but this is because DBSCAN forms **one big cluster**, which artificially stabilizes distances without true separation.

[20]:
```
# Cell: Heatmap of Noise Points for eps × min_samples (fixed)

import seaborn as sns
import matplotlib.pyplot as plt

# Pivot for heatmap
noise_heatmap = sil_df.pivot(index="min_samples", columns="eps",␣
 ↪values="noise_points")

plt.figure(figsize=(12, 6))
sns.heatmap(noise_heatmap, annot=True, fmt=".0f", cmap="magma")
plt.title("Noise Points Heatmap (eps vs min_samples)")
plt.xlabel("eps")
plt.ylabel("min_samples")
plt.show()
```



1. When **eps is very small (0.10–0.14)**, DBSCAN marks many points as noise because the neighborhood radius is too tight to form dense regions.
2. As **eps increases (0.18–0.26)**, noise sharply decreases because points begin to fall within each other's neighborhoods.
3. For **medium to large eps (0.28–0.60)**, noise becomes **zero**, meaning nearly all points get assigned to clusters.
4. Higher **min_samples** also reduces noise because only stable dense regions remain, improving cluster consistency.

[21]:
```
# Cell: Heatmap of Number of Clusters for eps × min_samples

import seaborn as sns
import matplotlib.pyplot as plt

# Pivot for heatmap
cluster_heatmap = sil_df.pivot(index="min_samples", columns="eps",␣
 ↪values="clusters_found")

plt.figure(figsize=(12, 6))
sns.heatmap(cluster_heatmap, annot=True, fmt=".0f", cmap="coolwarm")
plt.title("Clusters Found Heatmap (eps vs min_samples)")
plt.xlabel("eps")
plt.ylabel("min_samples")
plt.show()
```



- **High cluster count** = eps too small (over-splitting).
- **2 clusters** = eps in the optimal mid-range (correct result).
- **1 cluster** = eps too large (cluster merging).

This heatmap clearly shows how DBSCAN's parameters affect the number of clusters formed.

[ ]: